Grid DataBlade Programmer Guide

- Version 1.4 -

August 29, 2002

Table of Contents

INTRODUCTION 1

OVERVIEW 1
DATA IMPORT AND EXPORT 1
FUNCTIONALITY 1
SAMPLE APPLICATIONS 2

GRID-SPATIAL CONVERSION 2

OVERVIEW 2
THE GRID META-DATA 2
THE FORWARD PROCESS 4
THE REVERSE PROCESS 5
EXTRACTING GRID DATA 6
SPECIAL CASES 7
Step Sizes 7
Non-Uniform Grid Spacings 7
SAMPLING SCHEMES 8
Nearest-Neighbors 8
N-linear Interpolation 8

TECHNICAL MODELS 9

OPERATING MODEL 9
Client Assembling vs. Server Assembling 9
Data Assembling Model 9
DATATYPES 10

OPERATION 10

INSTALLATION AND CONFIGURATION 10
FILE-ORIENTED INTERACTION 10
Importing GIEF Files 11
Exporting GIEF Files 11
DIRECT OBJECT INTERACTION 12
Storing Data 12
Modifying Data 13
Extracting Data 14

APPLICATION PROGRAMMING INTERFACE (API) 15

SQL API LIBRARY 15
Examining a GRDValue 15
Modifying a GRDValue 15
Extracting a GRDValue 17

Additional Support Functions *17*
C API LIBRARY 18
Types and Constants: *19*
Creating and Setting a GRDValueI *19*
Examining the Contents of a GRDValueI *24*
Converting between a GRDValueI and a GRDValue (var binary) Type *28*
Creating and Setting a GRDSpecI Type *28*
Converting between a GRDSpecI and a GRDSpec (var binary) Type *31*
Example: Inserting Data Using ESQL/C *32*
Example: Extracting Data Using ESQL/C *33*
JAVA CLIENT API 36
Class Descriptions *36*
Example: Inserting Data Using Java *36*
Example: Extracting Data Using Java *38*

APPENDIX A: EXAMPLES OF GRID-SPATIAL CONVERSION 39

DEFAULT GRID META-DATA 40
TRANSLATED GRID 41
SCALED GRID 42
COLUMN MAJOR SCAN ORDER 43
SCAN DIRECTION 44
ROTATION 45
NON-UNIFORM GRID SPACING 46
COMBINING NON-UNIFORM GRID SPACING WITH ROTATION 47
COMBINING NON-UNIFORM GRID SPACING WITH ROTATION AND TRANSLATION 48

APPENDIX B: EXAMPLES OF GRDVALUE AND GRDSPEC 49

ORIGINAL GRID BASE LOCATION AND SAMPLING LOCATIONS 49
GRDVALUE REPRESENTATION 49
GRDSPEC REPRESENTATION 51

APPENDIX C: GRID IMPORT-EXPORT FORMAT (GIEF) 51

FEATURES 51
LIMITATIONS 51
CONVENTIONS 52
Grid Size *52*
Supporting Grids that Wrap *52*
Mapping Projection *52*
Translation *53*
Affine Transformation *54*
Nonuniform Axes *54*
Variable-Specific Attributes *55*
Other Attributes *55*
Grid Variables *56*
MAPPING NAMES FROM GIEF TO THE DATABASE 56
AN EXAMPLE GIEF FILE (AS A CDL FILE) 57
SAMPLE SPATIAL REFERENCE TEXT 58

APPENDIX D: USING S-EXPRESSIONS TO SPECIFY GRID EXTRACTION 65

TERMS IN S-EXPRESSIONS UNDERSTOOD BY THE GRID DATABLADE 65
EXAMPLE OF AN S-EXPRESSION 66
FORMAL S-EXPRESSION DEFINITION 67

APPENDIX E: ERROR MESSAGES 68

USER ERROR MESSAGES 68
PROGRAM FAILURE ERROR MESSAGES 69

Introduction

This document describes the Grid DataBlade, an Informix release 9.x server extension developed by Barrodale Computing Services to store and retrieve gridded data. The intended readership for this document is Grid DataBlade client application programmers.

Overview

The Grid DataBlade is an Informix Dynamic Server (IDS) extension that supports the storage, update, and fast retrieval of scalar data that has been sampled at each point of a rectilinear grid of two, three, or four dimensions. Two of the dimensions may be associated with the planar axes of a geographic projection, and each point of gridded data may have more than one value associated with it.

The Grid DataBlade uses an internal tiling scheme that reduces the number of disk seeks while being transparent to the caller, aside from the ability to specify tiling sizes. It also processes query results on the server, minimizing the amount of network I/O and client-side CPU time required.

Data Import and Export

The import/export format used by the Grid DataBlade is Grid Import-Export Format (GIEF). GIEF is a NetCDF dialect that is general enough to represent any grids of primitive scalar data elements (i.e., 1, 2 or 4-byte integers, 4 or 8-byte floating point values). It allows source grid files of various formats to have one internal uniform representation. Conversion programs can be employed to convert grid files between other formats (e.g., GeoTiff, GRIB, and dialects of NetCDF) and one or more GIEF files. The GIEF format is described in detail in this report (it should be noted that a separate GIEF standard, called GIEF-F, has been produced independently by FNMOC[1]).

Functionality

In addition to simple storage and retrieval, the Grid DataBlade supports:

- interpolating data between grid points using nearest neighbor or n-linear interpolation;
- extracting data in a different geographic projection from that used to create it;
- specifying particular axes as having non-uniform grid steps;

[1] Fleet Numerical Meteorology and Oceanography Center, Monterey, CA.

- storing multiple datasets (as many as 10) per grid, allowing several values of different types to be stored at a single point;
- storing and extracting data along arbitrary basis vectors rather than along the axes of a projection.

Sample Applications

Some possible applications for the Grid DataBlade are as follows:

- **Geographical**: for mapping a simple area of land, a 2D grid would suffice. Altitude may be stored in a field along with ground type (i.e. water, rock, or sand) and other important terrain features.

- **Geological**: a 3D grid may be used to store seismic data samples from geological surveys with fields such as density and water permeability. Samples may be non-uniformly spaced to provide more precise measurements near the surface.

- **Meteorological**: statistics on wind speed and direction may be input and analyzed in terms of location (x, y and z) and time. The x and y vectors of wind speed may be stored as two different fields in the same grid (as opposed to different rows in the same table).

- **Medical**: gridded data is not limited to large-scale samples taken from the earth and environment. It may also be used to map characteristics of the human body, for example x-ray results or visual data. One demonstration of the Grid DataBlade (see www.barrodale.com) uses data from the US National Library of Medicine Visual Human Project to extract a two-dimensional slice (oriented at arbitrary angles) from a three-dimensional dataset of the human body stored as a grid.

Grid-Spatial Conversion

Overview

This section provides a description of the concepts involved in the extraction of grid information using the Grid DataBlade, and provides descriptions of the parameters and processes involved in the conversion between spatial and grid coordinates. Examples of grid-spatial conversion are provided in Appendix A.

The Grid Meta-Data

Meta-data (i.e., data describing the data actually stored in the grid) is associated with each grid. This meta-data plays a key role in the conversion between grid and spatial coordinates. In the descriptions below, upper-case strings denote the keywords

associated with the meta-data, which, where appropriate, are also defined in terms of mathematical notation.

Each 4D[2] grid contains the following meta-data:

- **SRID**: spatial reference system id recognized by Spatial DataBlade.
- **FIELD NAMES and FIELD IDS**: textual names and integer ids for each dataset stored in the grid.
- **DIMENSION NAMES**: textual names for the grid axes, allowing names like "pressure" or "level".
- **STARTPT**: the location of the first element (in the specified spatial reference system). This can be represented mathematically as a 4-element column vector **s**, i.e.,

$$\mathbf{s} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ s_N \end{bmatrix}.$$

- **BASIS**: a set of four row vectors used to support the implementation of basis vectors. This can also be represented mathematically as an *4*x*4* matrix **V**, where each row represents a basis vector for the corresponding dimension, i.e.,

$$\mathbf{V} = \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} \\ v_{21} & v_{22} & v_{23} & v_{24} \\ v_{31} & v_{32} & v_{33} & v_{34} \\ v_{41} & v_{42} & v_{43} & v_{44} \end{bmatrix} \begin{matrix} \rightarrow BASIS\,0 \\ \rightarrow BASIS\,1 \\ \rightarrow BASIS\,2 \\ \rightarrow BASIS\,3 \end{matrix}.$$

  Note here that the keyword suffix range is from 0 to 3, while the subscript ranges are from 1 to 4.

- **NONUNIFORM**: a set of four vectors used to support irregularly spaced grids. This can also be mathematically represented as a 4x*M* matrix **U**, where *M* is the length of the longest dimension (by analogy to BASIS above, each row represents a vector), i.e.,

[2] 2D or 3D data is represented by a 4D grid with certain dimensions being of size 1.

$$\mathbf{U} = \begin{bmatrix} u_{11} & u_{12} & \mathrm{K} & u_{1M} \\ u_{21} & u_{22} & \Lambda & u_{2M} \\ u_{31} & u_{32} & \Lambda & u_{3M} \\ u_{41} & u_{42} & \Lambda & u_{4M} \end{bmatrix} \begin{matrix} \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \end{matrix} \begin{matrix} NONUNIFORM\,0 \\ NONUNIFORM\,1 \\ NONUNIFORM\,2 \\ NONUNIFORM\,3 \end{matrix}$$

Note: It is not necessary that the various dimensions have the same lengths; therefore, it is expected that some values be missing in the matrix defined above. These values will be denoted by *X*'s, and will not be used since they are out of range. For example, we can represent a NONUNIFORM matrix **U** for a 2x3x1x1 grid by the following:

$$\mathbf{U} = \begin{bmatrix} u_{11} & u_{12} & X \\ u_{21} & u_{22} & u_{23} \\ u_{31} & X & X \\ u_{41} & X & X \end{bmatrix}.$$

- **TILESIZE**: a set of values specifying how the grid is stored into memory. This can be represented by an 4-element column vector **t**, each element of which represents the length of the tile in the corresponding dimension, i.e.,

$$\mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix}.$$

With the exception of TILESIZE, all of these meta-data items play a role in determining how spatial coordinates are computed from grid positions, and vice versa. Each set of values determines the effect of one stage in the process of computing the spatial coordinates.

It is noted here that data is stored in row major order by the Grid DataBlade, as is the convention used in C and C++. (In contrast, FORTRAN uses column major ordering.)

The Forward Process

This process transforms a specified set of grid coordinate (i.e., location with respect to the grid) to their corresponding spatial coordinates. Assuming we have a 4D set of grid coordinates called GRIDCOORD (denoted by the 4-element column vector **g**), we compute the spatial coordinates SPATIALCOORD (denoted by the 4-element column vector **p**) as shown in the following steps. (In the expressions below, **a** and **b** are

temporary 4-element column vectors of coordinates, introduced so that the process can be broken into discrete steps.) Hence,

$$\mathbf{g} = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \\ g_4 \end{bmatrix}; \qquad \mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}; \qquad \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}; \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b \\ b_3 \\ b_4 \end{bmatrix}.$$

Step 1: Apply the non-uniform grid spacing.

For each dimension *i* (from 1 to 4)

$a_i = u_{i,g_i}$.

Step 2: Apply the basis vectors.

b = **Va**.

Step 3: Offset by the start point.

p = **b** + **s**.

The above three steps can be described more tersely as a function that maps (i0,i1,i2,i3) to a new tuple defined by the expression:

p = **Va** + **s**.

The Reverse Process

This process transforms a specified set of spatial coordinates (i.e., location in space) to the corresponding grid coordinates. It is approximately the reverse of the forward process.

Step 1: Relocate the first element to the origin for the grid.

b = **p** – **s**.

Step 2: Remove the effect of the basis vectors.

a = $\mathbf{V}^{-1}$**b**.

Note: While **V** need not be orthogonal, it must be invertible for the reverse process to succeed.

Step 3: Remove the effect of the non-uniform grid spacing (assuming nearest neighbors).

For each dimension i (from 1 to 4)
Determine the maximum value of j such that $1 \le j \le M$ and $u_{ij} \le a_i$
$g_i = j + (a_i - u_{ij})/(u_{i,j+1} - u_{ij})$

Note: As previously mentioned, M represents the number of columns in the matrix **U**. There is also an implicit constraint that the each row of **U** must contain strictly increasing values if the reverse process is to succeed.

Extracting Grid Data

Extracting data from a grid involves both the forward process and the reverse process. Conceptually, the extraction process involves the formation of a destination grid from a stored source grid, and operates as follows:

For each grid position in the destination grid:

- Apply the forward process to the destination grid position to compute the spatial coordinate corresponding to that position.
- If necessary, re-project that spatial coordinate from the destination grid's projection to the source grid's projection.
- Apply the reverse process to the re-projected spatial coordinate to compute the corresponding position **g** (GRIDCOORD) in the source grid.
- Compute a value from the source grid based on this position in the source grid.
- Store that value in the destination grid at the destination grid position.

The treatment of non-integer values in **g** (GRIDCOORD) depends on the nature of the interpolation chosen. The nearest neighbor scheme rounds these values to the nearest integer, while linear interpolation uses the fractional components of the positions to compute a weighted average (see Section "Sampling Schemes" below).

Another item that needs explanation is the order of the spatial coordinates. First of all, this order is immaterial as long as the application extracting the data uses the same projection and ordering as the application that stored the data. The act of projecting the coordinates, however, is a non-linear function which makes very strong assumptions about which part of a spatial coordinate tuple is the x coordinate, and which is the y coordinate. For this reason, spatial coordinates are always listed in the order (x, y, z, t) and the row vectors for BASIS are defined consistent with that ordering.

Note that because the reverse process is performed on source grid meta-data, it is necessary that the source grid meta-data have an invertible BASIS (i.e., **V**) and strictly increasing values in the rows in the NONUNIFORM array (i.e., **U**). These requirements do not apply for the destination grid.

Special Cases

Step Sizes

Step sizes are closely related to the basis vectors described above. In fact, the functionality that appears externally as step sizes is actually just a wrapper around the basis vectors mechanism. Specifying step sizes actually produces a set of basis vectors that form a diagonal matrix with the step sizes along the diagonal. For example, specifying step size values (3, 4, 9, 1) is converted to the following basis vectors:

BASIS 0: 3 0 0 0
BASIS 1: 0 4 0 0
BASIS 2: 0 0 9 0
BASIS 3: 0 0 0 1

or mathematically, $\mathbf{V} = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$.

Non-Uniform Grid Spacings

Most applications do not require non-uniform grid spacings. The Grid DataBlade treats the absence of a specified non-uniform grid spacing array as being equivalent to a non-uniform grid spacing array containing the values $\{0, 1, 2, 3, ..., M_i-1\}$ where M_i is the number of grid positions in the ith dimension. For instance, if we had not specified non-uniform spacings along the second dimension (i.e., NONUNIFORM 1) when specifying non-uniform spacings for a 2x3x4x5 grid with, for example,

NONUNIFORM 0: 2 3
NONUNIFORM 2: 0 2 9 11
NONUNIFORM 3: 0 1 2 5 10

the missing non-uniform spacing (i.e., along the second dimension) would have been interpreted as:

NONUNIFORM 1: 0 1 2 (since the length of the second dimension is 3).

Also note that we can alternatively represent this mathematically by the following matrix: (*X*'s are used to represent missing values as the three dimensions are of unequal lengths).

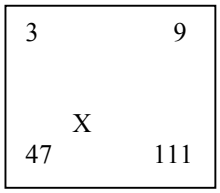
$$\mathbf{U} = \begin{bmatrix} 2 & 3 & X & X & X \\ 0 & 1 & 2 & X & X \\ 0 & 2 & 9 & 11 & X \\ 0 & 1 & 2 & 5 & 10 \end{bmatrix} \begin{matrix} \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \end{matrix} \begin{matrix} \textit{NONUNIFORM } 0 \\ \textit{NONUNIFORM } 1 \\ \textit{NONUNIFORM } 2 \\ \textit{NONUNIFORM } 3 \end{matrix}.$$

Sampling Schemes

When data is extracted from a source grid, there may not be an exact match between grid cell locations in the extracted grid and in the sampled grid. This may be caused by any combination of the following:

- the start point of the extracted grid does not fall on a source grid cell location,
- the basis vectors of the two grids are not equal, or
- the SRIDs are different

When there is not an exact match between grid coordinates, as shown in the figure below, one of two sampling methods is used, as described in the following sections.

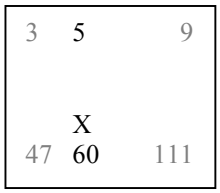


Nearest-Neighbors

The simplest method is *nearest-neighbors*. Nearest-neighbors is conceptually equivalent to snapping to the nearest grid point. In the example shown above, the extracted grid value at X would be assigned the value 47 because X is closest to the grid cell containing 47.

N-linear Interpolation

The other method is N-linear interpolation. N-linear interpolation is a simple generalization of bi-linear interpolation. In the 2-D case shown, one would first linearly interpolate along one axis, as shown below:



and then linearly interpolate the interpolated values (5, 60) to give a value of 50 at X.

Technical Models

Operating Model

Client Assembling vs. Server Assembling

In a traditional database environment, custom functionality is implemented purely on the client side, resulting in "fat clients". IDS release 9.x allows user-defined routines (UDRs) in C, which provides a programmer much greater control over where functionality resides. Data can be assembled/disassembled on either the client or the server, or both, with attendant tradeoffs, as in the following table.

Pros to assembling on the client	Pros to assembling on the server
• An individual client is easier to debug and has fewer limitations tied to issues of thread safety, blocking, and legacy code. • A client typically uses a single-threaded model, allowing a wider variety of system functions to be safely used. • Very large data objects can often be handled analogously to file processing operations, reducing the memory needs (i.e., the entire data object might not need to be in memory at any point in time).	• It is possible to take advantage of server support for handling byte ordering and other language or platform issues. • Centralization of the code reduces the amount of logic linked into thinner clients, shrinking the possibility of inconsistencies between different clients. • Reduces network I/O by transmitting just the query results to the client with a minimum of synchronization issues. • Provides better data integrity through concurrency control, transaction management, backup, and recovery.

Which model to use depends a great deal on the size of the datatypes, the nature of the custom functionality, and the nature of the data accesses.

Data Assembling Model

The Grid DataBlade uses the approach of assembling on the server side. This makes it ideal for centralized databases that will be accessed from abroad, particularly through slow connections or where data transfer speed is an important issue. The primary reason for this is that only the requested results are moved across the client/server connection. For example, extracting data at five grid points causes those five points to be pulled

across the connection, and does not include all the internal tiles that contain the five data points. This poses a definite advantage when data is sub-sampled.

Datatypes

Underlying the Grid DataBlade are two basic opaque types:

- **GRDValue**: stores a 4D grid of user data along with the earlier mentioned meta-data for Grids.
- **GRDSpec**: contains information needed to extract a specific dataset from a GRDValue. Information found in the GRDSpec over-rides the meta-data found in the GRDValue being extracted from. This may include none, some of, or all of the meta-data stored in a grid in addition to the following properties:
  - **SRTEXT**: A GRDSpec may specify SRTEXT instead of SRID. Specifying the SRTEXT is more general than using the SRID since the latter provides for specifying a range of valid values or coordinate precisions. Hence, the SRTEXT is more likely to match the SRID of the GRDValue from which the data is to be extracted
  - **INTERPOLATION**: a specification, for each grid axis, of whether nearest-neighbor or N-linear interpolation should be used.

Examples of GRDValue and GRDSpec are given in Appendix B.

Operation

Installation and Configuration

The Grid DataBlade may be installed and configured by the following steps:

1. Create a directory in $INFORMIXDIR/extend/ called Grid.1.0.0.0 with owner and group Informix.
2. Copy the distribution files into that directory.
3. Remove group/other write permissions from the Grid.bld file if they are set.
4. Use bladeMgr to install the DataBlade in any desired databases.
5. Create a directory in /tmp called grid_temp. Change its permissions so that all users (most importantly, the Informix server process) can write into that directory.

File-Oriented Interaction

The Grid DataBlade is also able to read and write a subset of NetCDF files directly on a client machine, simply through the execution of the appropriate SQL statement. The statements can be executed by a user-written client program (using JDBC, ODBC, or ESQL/C) or simply by statements entered into any command line based database tool,

(such as the Informix-supplied dbaccess program). The subset of NetCDF supported is a dialect called GIEF (Grid Import-Export Format). GIEF is described in more detail in Appendix C.

Note: in order for the functions below to operate, there must be a server-writeable directory on the host machine called /tmp/grid_temp.

Importing GIEF Files

A GIEF file is imported by executing the GRDFromGief routine. This routine has the following definition:

```
create procedure GRDFromGief(
        lvarchar, -- file name
        lvarchar, -- table name
);
```

The file name is a text string giving the path (either relative or absolute) of the file name to be read. The file must be readable by the server. The table name must be the name of an already existing table with the appropriate set of columns to handle the attributes in the GIEF file.

Exporting GIEF Files

A GIEF file is exported by executing the GRDRowToGief routine. This routine has the following definition.

```
create function GRDRowToGief(
        lvarchar, -- file name
        lvarchar, -- table name
        integer,  -- the rowid of the row of in the table.
        GRDSpec, -- specification of what to extract
) returns integer;
```

The file name must be the path of the file to be written. The table name is the name of the table from which to extract the grid. The row id is the integer value of the row-id ‘column’, a special column added by Informix automatically for most tables. The GRDSpec indicates which fields and what portions of them to extract from the specified grid.

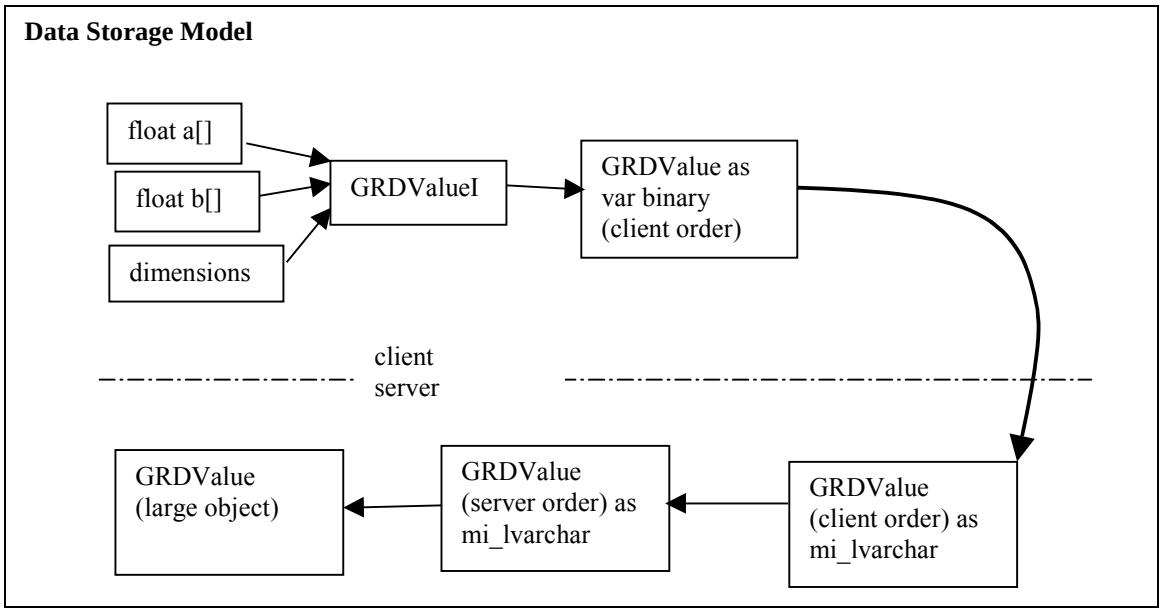
When calling GRDRowToGief, it is simplest to express the GRDSpec as a text string and then cast it to GRDSpec via the double-colon operator (this avoids having to link in any special libraries). The text form of a GRDSpec is based on S-expressions and is provided in Appendix D.

Direct Object Interaction

Note: this section can be ignored by application writers who are only interested in GIEF files for I/O.

Storing Data

For the Grid DataBlade, we have chosen the following model for storing data (as shown in the diagram below):

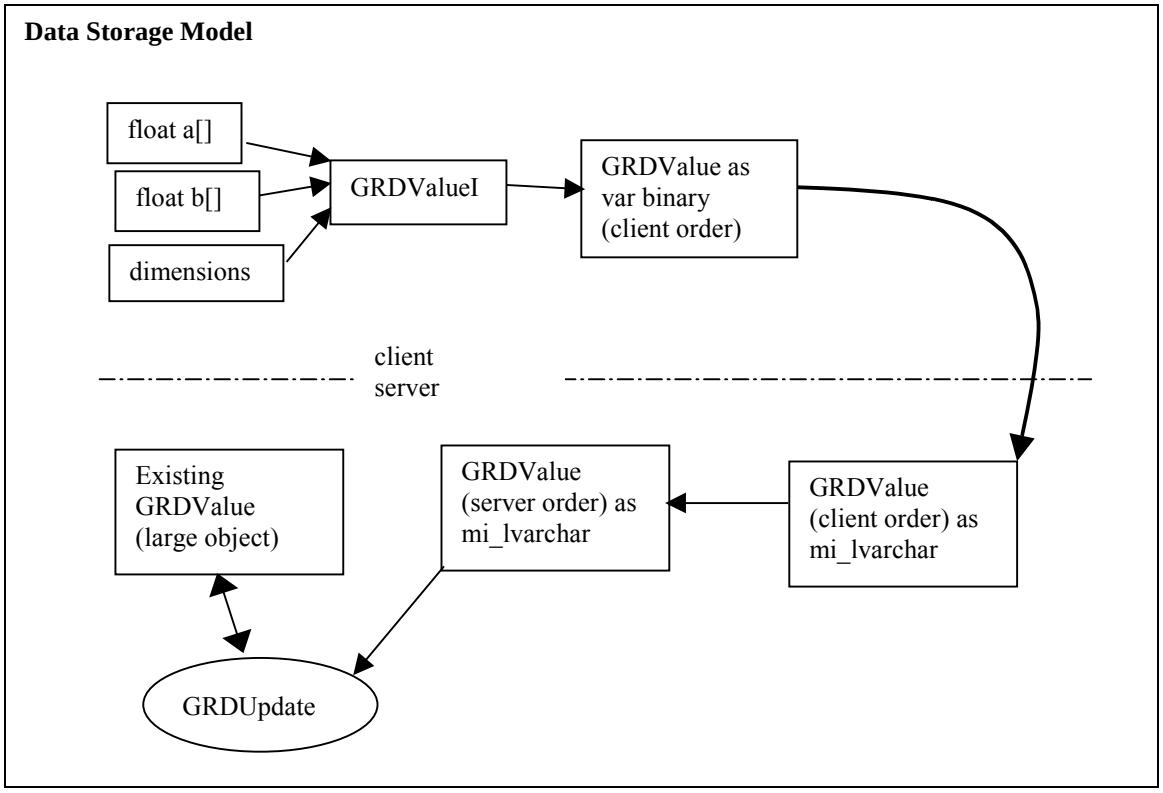


Notes:

1. A client program creates a data structure called a GRDValueI and populates it with values from its local arrays. The GRDValueI is a straightforward C data structure containing pointers to several variable length arrays.
2. The client then converts the GRDValueI to a GRDValue stored inside a var binary. The GRDValue is an alternate representation of the GRDValueI, one in which all the data exists in a contiguous area of memory and does not contain any pointers. A var binary is a variable length binary structure used to hold variable length binary data.
3. The ESQL/C API is used to ship the var binary data from client to the server, which repacks the var binary structure as an mi_lvarchar structure.
4. The server produces a new version of the GRDValue that has the server's byte order.
5. A GRDValue assign support function causes the GRDValue to be stored as a small object containing a smart blob.

Modifying Data

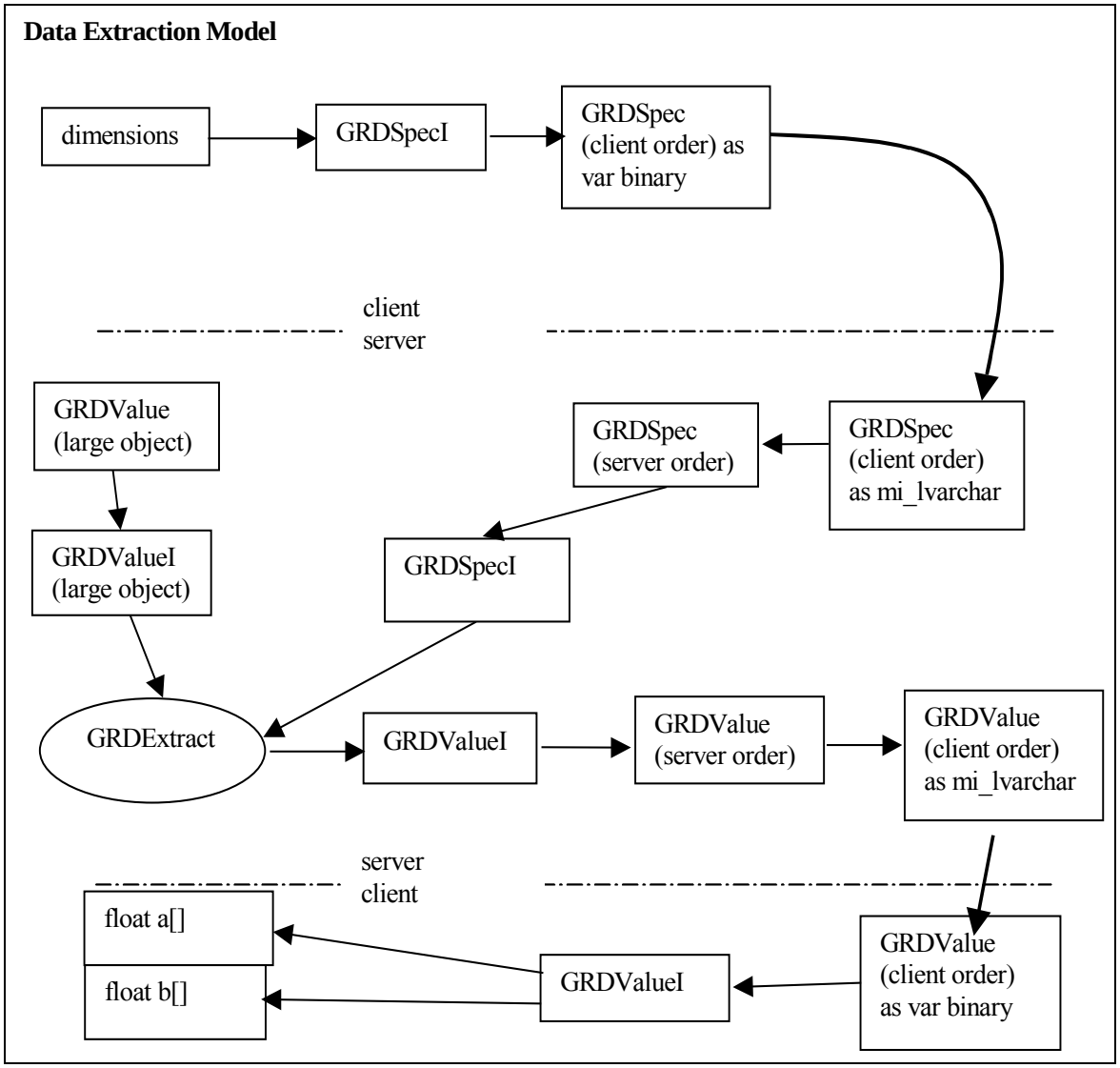
Modifying a GRDValue is identical to inserting a GRDValue into a table except for the small detail that there is one GRDValue being modified, and another GRDValue serving as the source of the modification data. The actual modification is done with a simple update statement. See the SQL API GRDUpdate for more details.



It should be noted that once a GRDValue has been stored in a database, only its grid values can be modified, not its meta-data. For instance, you cannot change the GRDValue's start point, number of dimensions, the number of samples in any dimension, the number or type of the fields.

Extracting Data

To extract data, a similar model is used (as shown in the diagram below):



Notes:

1. A GRDSpecI data structure is created and populated by a client application. Like the GRDValueI, the GRDSpecI is a straightforward C data structure with pointers to handle variable length components.
2. The GRDSpecI is converted to a GRDSpec (an alternate representation that uses a contiguous block of memory and no pointers).
3. Through the ESQL/C API, the GRDSpec gets passed to the server, which constructs a new version with the server's byte order.

4. The GRDExtract UDR on the server gets called with the GRDSpec and a GRDValue to extract from.
5. Internally, the GRDExtract UDR unpacks the GRDSpec into a GRDSpecI, the header of the GRDValue into a GRDValueI, and builds a new GRDValueI, converts the new GRDValueI into a GRDValue, and returns the GRDValue.
6. The server computes a new GRDValue that has the client's byte order.
7. The client program receives the GRDValue from the server, unpacks it into a GRDValueI, and copies the contained data into local arrays.

Application Programming Interface (API)

SQL API Library

The following SQL API is intended to be used in the course of developing applications. The SQL user-defined routines (UDRs) below deal with storing data into a database and extracting subsets of that data. The SQL API is meant to be used in conjunction with the C API or the Java API, which deal with the datatypes used for application programming.

Examining a GRDValue

function GRDResolution(GRDValue,integer) returns double precision: Returns the resolution along one of the grid dimensions, where the second argument is the dimension number (1-4). The function returns the distance between consecutive grid points along that dimension's axis. The values along a grid axis that crosses spatial and non-spatial coordinates is, of course, rather meaningless.

function GRDSrid(GRDValue) returns integer: Returns the SRID of the projection system the GRDValue is in. Given the SRID, the spatial reference text can be retrieved from the spatial_references table.

function GRDToGeodetic(GRDValue) returns lvarchar: returns the text for a GeoPolygon, GeoLineseg, or GeoPoint that surrounds the Grid, depending on 2D dimensionality of the grid. The text can then be cast to a GeoObject, and in practice just about all the grids can be cast to a GeoPolygon. This function is present to support geospatial indexing.

function GRDStartPoint(GRDValue) returns lvarchar: this function returns the start point (i.e., translation) of a grid as a text string of space separated values.

Modifying a GRDValue

function GRDExtend(original GRDValue, sexpr lvarchar) returns GRDValue: this function is used to clone a GRDValue while enlarging some of its dimensions. The sexpr

is a text string containing an S-expression. Currently, the only terms allowed in sexpr are a final_size term. Any of the below are valid values for sexpr:

'(final_size 100 100 1 1)'
'((final_size 1 1 100 100))'
'()'

The final_size term allows a GRDValue of a particular size to made into grid of larger dimensions. The size of the dimension in the new grid is the larger of the `suggested size' and the original size. This means that the first expression above would expand the first and second dimensions of a grid to 100 and 100 while leaving the third and fourth dimension unchanged. The second expression would expand the third and fourth dimensions while leaving the first two dimensions unchanged.

There is one constraint on this function (besides the dimensions being positive): The data in the original grid must occupy a contiguous region of blob space in the returned grid. The rules that preserve this constraint are:

- size[i] cannot be expanded if i-t >= 1 and has a value > 1 for any t >= 1.
- size[i] cannot be expanded if i+t <= 4 and size[i+t] > 1 and size[i+t] is going to be expanded for any t >= 1.

function GRDUpdate(original GRDValue, mods GRDValue) returns GRDValue: this UDR modifies the first GRDValue (original) to add the new information contained in the second GRDValue (mods).

GRDUpdate is always called implicitly, by executing the update statement, in the form:

```
update myTableA set myColumnA = some_grdvalue_expression
where some_column_name = some_value ;
```

There are a number of constraints on the two GRDValues:

- **mods** and **original** must have the same SRID and the same set of basis vectors.
- **mods** can only contain fields whose name and type match those fields in **original**.
- for any dimension that does not have a nonuniform dimension **mods** data must fit entirely inside **original**, before grid wrap-around is considered.
- for any dimension that has a nonuniform dimension, the positions in the **mods** data must either match those in **original**, or be greater than the last entry so that the **mods** dimension positions can be stacked after the **original** positions for that dimension.
- the data elements in **mods** must occupy a contiguous region of blob space in **original**. This is to limit the number of log entries generated. The following sets of dimensions (one set per row of the table) are accepted:

mods size[0]	mods size[1]	mods size[2]	mods size[3]
<= original size[0]	= original size[1]	= original size[2]	= original size[3]
1	<= original size[1]	= original size[2]	= original size[3]
1	1	<= original size[2]	= original size[3]
1	1	1	= original size[3]

Extracting a GRDValue

function GRDExtract(source GRDValue, spec GRDSpec) returns GRDValue: this UDR creates and returns a new GRDValue that contains information taken from the source GRDValue parameter in regions specified by the GRDSpec parameter, in the projection specified by the SRID parameter.

This UDR will usually be called from client applications to extract data, in the following form. Note: the terms preceded by colons represent host variables in ESQL/C programs.

```
select GRDExtract(myColumn, :myGRDSpec)
from myGridTable where GRDId(origValue) = :myTargetId;
```

function GRDExtract2Pass(source GRDValue, spec GRDSpec) returns GRDValue: this UDR has identical semantics to GRDExtract but a different performance profile. Rather than fetching tiles from a source grid as needed (as GRDExtract does), it first computes which tiles it will need, reads the tiles in the order that they are found in the GRDValue, and then builds the new GRDValue. With the current implementation, this doubles the amount of CPU time required but can substantially reduce the disk I/O time, since:

- the disk's track head seeks in only one direction (not back and forth as it would for some extractions);
- the disk is used for a shorter time frame per extraction, decreasing the likelihood that simultaneous queries will interfere with each other;
- fewer disk reads are performed, as tiles that are adjacent on the disk can be read in a single disk read operation.

The exact nature of performance speed-up (or slow-down as the case may be) depends heavily on the respective speeds of the processors versus the hard-drives, the appropriateness of the tile sizes, and the amount of work involved in the extract. Faster processors, slower hard-drives, and extracting data in the same projection as it was stored tend to give GRDExtract2Pass an advantage over GRDExtract. A general rule of thumb is that if the CPU utilization is less than 35% when using GRDExtract, then GRDExtract2Pass is likely to give greater performance.

Additional Support Functions

function GRDTrim(blankPaddedText lvarchar) returns lvarchar: this function is capable of trimming white space (preceding and following) from a text string. The standard SQL trim function does not operate on the lvarchar data type and is limited to

strings of up to 256 characters long. This function is useful when trying to compare a literal string to the srtext entry of the spatial_references table.

procedure GRDGenSrid(srtext lvarchar): this procedure generates an entry in the spatial_references table for a particular instance of spatial reference text, if that entry has not already been created. It is occasionally needed before running GRDRowToGief[3]. It establishes its own transaction, so it must not be run from inside a transaction. The srtext may have (@) signs in place of embedded double quotes, following the convention for srtext in S-expressions.

function GRDIsWholeSphere(grid GRDValue) returns boolean: determines whether a grid represents the entire sphere or not. Currently, this function only handles geographical coordinate systems (i.e., lat/long); it will always return false for a projected coordinate system.

function GRDSecsToTime(timeInSeconds integer) returns lvarchar: returns the textual representation of the broken down time in datetime format (year to seconds) represented by the integer time in seconds. This is useful in formulating geodetic time queries.

C API Library

The following C API is intended to be used by ESQL/C application developers in the course of developing applications. It is meant to be used in conjunction with the SQL API in the preceding section.

Notes:

- The C API occasionally uses mi_integer and other mi_types, which are used to guarantee the same number of bits on different architectures. These types are defined in the Informix include file mitypes.h.
- Functions for creating and viewing the contents of a GRDValueI, creating a GRDSpecI, converting between a GRDValueI and a GRDValue, and converting a GRDSpecI to a GRDSpec are provided and described in this section. Functions for viewing the contents of a GRDSpecI and converting a GRDSpec to a GRDSpecI are available in the C API library but are not described here, since they have no role in writing client applications.

[3] Data can only be projected to mapping projections described by the spatial_references table. These mapping projections can only be added while inside a transaction. A transaction cannot be started while executing a select statement. Most uses of the GRDRowToGief function are done within a select statement. This prevents the GRDRowToGief function from being able to add elements to the spatial_references table itself.

Types and Constants:

```
#define maxGrdDIMENSIONS (4)
#define maxGrdFIELDS (10)

typedef enum {
  grdFieldUNSET = 0,    /* no value set */
  grdFieldINT1,         /* 1 byte signed integer */
  grdFieldINT2,         /* 2 byte signed integer */
  grdFieldINT4,         /* 4 byte signed integer */
  grdFieldINT8,         /* 8 byte signed integer */
  grdFieldUINT1,        /* 1 byte unsigned integer */
  grdFieldUINT2,        /* 2 byte unsigned integer */
  grdFieldUINT4,        /* 4 byte unsigned integer */
  grdFieldUINT8,        /* 8 byte unsigned integer */
  grdFieldREAL4,        /* 4 byte float */
  grdFieldREAL8,        /* 8 byte float */
  grdFieldRGBA,         /* 4 1 byte values */
  grdFieldPOLAR4,       /* 2D vector as theta/r 4 byte float pairs */
  grdFieldPOLAR8,       /* 2D vector as theta/r 8 byte float pairs */
  grdFieldSPHERICAL4,   /* 3D vector as theta/sigma/r 4 byte triplets */
  grdFieldSPHERICAL8,   /* 3D vector as theta/sigma/r 8 byte triplets */
  grdFieldMustBeLast    /* number of field specifications */
} GRDFieldType;
```

Creating and Setting a GRDValueI

The GRDValueI is an intermediate structure used to construct the GRDValue. The following functions are used to construct the GRDValueI.

```
/*
 * Name: GRDValueINew
 * Purpose: Allocate a new GRDValueI structure
 * Returns: the new structure.
 */
GRDValueI *GRDValueINew();

/*
 * Name: GRDValueISetDims
 * Purpose: Establishes the number of dimensions held by the GRDValueI
 * Arguments:
 *    p: the GRDValueI
 *    numDims: the number of dimensions involved.
 *    numSamples: number of samples in each dimension
 *    axisIds: name of each dimensional axis
```

```
 * Comments: you must establish the dimensions before defining fields
 *    since memory allocations for the fields depend on the dimensions.
 */
void GRDValueISetDims(GRDValueI *p, mi_integer numDims, mi_integer
numSamples[], mi_integer axisIds[]);

/*
 * Name: GRDValueISetStartPt
 * Purpose: set the location of the first element stored in the grid.
 * Arguments:
 *   p: the GRDValueI
 *   startPt: an array of doubles
 * Comment: the number of dimensions must be set first.
 */
void GRDValueISetStartPt(GRDValueI *p, mi_double_precision startPt[]);

/*
 * Name: GRDValueISetBasisVector
 * Purpose: set the basis vector for the ith dimension.
 * Arguments:
 *    p: the GRDValueI
 *    dimNum: the index of the dimension being set
 *    BasisVector: vector defining the orientation of the dimension
 *       relative to the native coordinate system.
 */
void GRDValueISetBasisVector(GRDValueI *p, mi_integer dimNum,
mi_double_precision BasisVector[]);

/*
 * Name: GRDValueISetAligned
 * Purpose: sets the grid step vectors for dimensions for the simple
 *    case that the ith dimension is aligned along the ith
 *    axis.
 * Arguments:
 *    p: the GRDValueI
 *    stepValues: the step value along each axes of the native
 *        coordinate system.
 * Comments: this is an alternative to using GRDValueISetIthDim.
 */
void GRDValueISetAligned(GRDValueI *p, mi_double_precision stepValues[]);

/*
 * Name: GRDValueISetNonUniform
 * Purpose: define a nonuniformly sampled dimension
 * Arguments:
 *   p: the GRDValueI
```

```
 *   dimNum: the number of the dimension that should be nonuniform
 *   samplePoints: where to sample along the dimNumth dimension.
 *     These values represent scaling factors applied to the dimensions
 *     stepping vector.
 */
void GRDValueISetNonUniform(GRDValueI *p, mi_integer dimNum,
mi_double_precision samplePoints[]);

/*
 * Name: GRDValueISetNumFields
 * Purpose: sets the number of data fields that can be stored.
 * Arguments:
 *   p: the GRDValueI
 *   numFields: the number of fields p should have.
 */
void GRDValueISetNumFields(GRDValueI *p, mi_integer numFields);

/*
 * Name: GRDValueISetFieldType
 * Purpose: sets the parameters describing one of the fields
 * Arguments:
 *   p: the GRDValueI
 *   fieldNum: which field to set (a value between 0 and numDataFields -1)
 *   fieldId: an integer id to be assigned to the field
 *   fieldType: the datatype of the field's elements (e.g., grdFieldREAL4).
 */
void GRDValueISetFieldType(GRDValueI *p, mi_integer fieldNum, mi_integer fieldId,
GRDFieldType fieldType);

/*
 * Name: GRDValueIAppendFieldType
 * Purpose: sets the parameters describing the next available field.
 * Returns: the index of the field.
 * Arguments:
 *   p: the GRDValueI
 *   fieldId: an integer id to be assigned to the field
 *   fieldType: the datatype of the field's elements (e.g., grdFieldREAL4).
 * Comments: this is an alternative to using GRDValueISetNumFields and
 *   GRDValueISetFieldType, useful when you do not know the number of
 *   fields in advance and consecutive ordering of the fields is sufficient.
 */
mi_integer GRDValueIAppendField(GRDValueI *p, mi_integer fieldId, GRDFieldType
fieldType);

/*
 * Name: GRDValueISetAlignTile
```

```
 * Purpose: causes the tiles to be aligned on disk page boundaries
 *    when stored in the database. The default is no alignment.
 * Arguments:
 *    p: the GRDValueI
 */
void GRDValueISetAlignTile(GRDValueI *p);

/*
 * Name: GRDValueISetAlignField
 * Purpose: causes the field components inside a tile to be aligned on
 *   disk page boundaries when stored in the database. The default is
 *   no alignment.
 * Arguments:
 *    p: the GRDValueI
 * Comments: this results in more internal fragmentation than
 *   using GRDValueISetAlignTile.
 */
void GRDValueISetAlignField(GRDValueI *p);

/*
 * Name: GRDCalcNumFieldEle
 * Purpose: computes the number of fields
 * Arguments:
 *   p: the GRDValueI
 * Returns: the number of elements as long long (64-bit quantity)
 */
long long GRDCalcNumFieldEle(GRDValueI *p);

/*
 * Name: GRDValueISetValues
 * Purpose: transfer data from a source array to the GRDValueI,
 * Arguments:
 *   p: the GRDValueI
 *   fieldNum: index of the field being set (not the id)
 *   data: point to the array of input data, assumed to match the
 *         previously defined field type.
 *   dataValid: an array of Boolean flags indicating whether a
 *         particular element of "data" is valid (!0) or null (0).
 *   startIndex: where in p to store the first element from "data". If null,
 *         then data is assumed to exactly fit the relevant field.
 *   sourceDims: dimensions of the "data" array. This should non-null if and only if
 *       startIndex is non-null.
 */
void GRDValueISetValues(GRDValueI *p, mi_integer fieldNum, void *data,
ElementSetFlag  *dataValid,  mi_integer startIndex[grdMaxDIMENSIONS], mi_integer
sourceDims[grdMaxDIMENSIONS]);
```

```
/*
 * Name: GRDValueISetTileSize
 * Purpose: set the sizes of the internal tiling.
 * Comment: tiling is an optimization issue. An appropriate tiling size
 *   increases performance only.
 * Arguments:
 *   p: the GRDValueI
 *   sizes: the dimensions of a tile
 */
void GRDValueISetTileSize(GRDValueI *p, mi_integer sizes[]);

/*
 * Name: GRDValueISetSrid
 * Purpose: Sets the projection associated with the GRDValueI
 * Arguments:
 *   p: the GRDValueI
*   SRID: the spatial reference id.
 * Comments: it is an error to try to associate a projection
 *   after data has already been stored or extracted.
 */
void GRDValueISetSrid(GRDValueI *p, mi_integer SRID);

/*
 * Name: GRDValueISetDimName
 * Purpose: sets the textual name of a field
 * Arguments:
 *    p: the GRDValueI
 *    dimNum: the index of the field to be given a name
 *    dimName: the textual name.
 * Comments: a copy is made of dimName, the parameters doesn't
 *    need to be preserved for the length of the object.
 */
void GRDValueISetDimName(GRDValueI *p, int dimNum, char *dimName);

/*
 * Name: GRDValueISetFieldName
 * Purpose: sets the textual name of a field
 * Arguments:
 *    p: the GRDValueI
 *    fieldNum: the index of the field to be given a name
 *    fieldName: the textual name.
 * Comments: a copy is made of fieldName, the parameters doesn't
 *    need to be preserved for the length of the object.
 */
void GRDValueISetFieldName(GRDValueI *p, int fieldNum, char *fieldName);
```

```
/*
 * Name: GRDValueISetWrap
 * Purpose: indicates that the grid wraps around in a particular dimension.
 * Arguments:
 *   p: the GRDValueI
 *   dimNum: dimension the wrapping applies to.
 */
void GRDValueISetWrap(GRDValueI *p, int dimNum);
```

Examining the Contents of a GRDValueI

```
/*
 * Name: GRDValueIGetValues
 * Purpose: transfer data from a source array to the GRDValueI,
 * Arguments:
 *   p: the GRDValueI
 *   fieldNum: index of the field being set (not the id)
 *   data: point to the array of input data, assumed to match the
 *         previously defined field type.
 *   dataValid: an array of Boolean flags indicating whether a
 *         particular element of "data" is valid (!0) or null (0).
 *   startIndex: where in p to store the first element from "data". If null,
 *         then data is assumed to exactly fit the relevant field.
 *   sourceDims: dimensions of the "data" array. This should non-null if and only if
 *        startIndex is non-null.
 */
void GRDValueIGetValues(GRDValueI *p, mi_integer fieldNum, void *data,
ElementSetFlag  *dataValid, mi_integer startIndex[grdMaxDIMENSIONS], mi_integer
sourceDims[grdMaxDIMENSIONS]);

/*
 * Name: GRDValueIGetSrid
 * Purpose: Gets the SRID associated with the GRDValueI
 * Arguments:
 *   p: the GRDValueI
* Comments: it is an error to try to associate a projection
 *   after data has already been stored or extracted.
 */
mi_integer GRDValueIGetSrid(GRDValueI *p);

/*
 * Name: GRDValueIGetStartPt
 * Purpose: get the location of the start point
 * Arguments:
```

```
 *   p: the GRDValueI
 * Returns: an array of doubles
 * Comments: the array is part of "p"; it only lives as long as "p"
 *   and should not be modified.
 */
mi_double_precision *GRDValueIGetStartPt(GRDValueI *p);

/*
 * Name: GRDValueIGetBasisVector
 * Purpose: get the basis vector for the ith dimension.
 * Arguments:
 *    p: the GRDValueI
 *    dimNum: the index of the dimension being set
 * Returns: the dimNumth basis vector (counting from 0)
 * Comments: this returns a pointer to an array inside of p. The array is only
 *    valid during the life of "p" and it is illegal to modify the array.
 */
mi_double_precision *GRDValueIGetBasisVector(GRDValueI *p, mi_integer
dimNum);

/*
 * Name: GRDValueIGetNumDims
 * Purpose: Returns the number of dimensions held by the GRDValueI
 * Arguments:
 *    p: the GRDValueI
 * Returns: the number of dimensions.
 */
mi_integer GRDValueIGetNumDims(GRDValueI *p);

/*
 * Name: GRDValueIGetDimensions
 * Purpose: returns the vector size of the grid
 * Arguments:
 *    p: the GRDValueI
 * Returns: an array of mi_integer
 * Comments: the array returned is part of p. It should not be modified.
 */
mi_integer *GRDValueIGetDimensions(GRDValueI *p);

/*
 * Name: GRDValueIGetAxisIds
 * Purpose: returns an array of Axis Ids.
 * Arguments:
 *    p: the GRDValueI
 * Returns: an array of mi_integer.
 * Comments: the array returned is part of p. It should not be modified.
```

```
 */
mi_integer *GRDValueIGetAxisIds(GRDValueI *p);

/*
 * Name: GRDValueIGetNumFields
 * Purpose: returns the number of fields represented by a GRDValueI
 * Arguments:
 *    p: the GRDValueI
 * Returns: the number of fields
 */
mi_integer GRDValueIGetNumFields(GRDValueI *p);

/*
 * Name: GRDValueIGetFieldId
 * Purpose: returns the id of a particular field in the GRDValueI
 * Arguments:
 *  p: the GRDValueI
 *  position: which field [0..GRDValueIGetNumFields(p)-1]
 * Returns: the field id.
 */
mi_integer GRDValueIGetFieldId(GRDValueI *p, mi_integer position);

/*
 * Name: GRDValueIGetFieldType
 * Purpose: returns the type of a particular field in the GRDValueI
 * Arguments:
 *   p: the GRDValueI
 *   position: which field [0..GRDValueIGetNumFields(p)-1]
 * Returns: the field type as a GRDFieldType enumeration
 */
GRDFieldType GRDValueIGetFieldType(GRDValueI *p, mi_integer position);

/*
 * Name: GRDValueIGetFieldData
 * Purpose: returns a pointer to the internal array used to
 *   hold data for a particular field in a GRDValueI
 * Arguments:
 *   p: the GRDValueI
 *   position: which field  [0..GRDValueIGetNumFields(p)-1]
 * Returns: a pointer to the internal array.
 * Comments: if the internal array has not been allocated yet,
 *   this call causes it to be allocated. It is the caller's responsibility
 *   to ensure that their use of the array is consistent with its dimensions
 *   and element type.
 */
void *GRDValueIGetFieldData(GRDValueI *p, mi_integer position);
```

```
/*
 * Name: GRDValueIGetFieldValid
 * Purpose: returns a pointer to the internal array used to
 *   track which elements in a field are valid.
 * Arguments:
 *   p: the GRDValueI
 *   position: which field  [0..GRDValueIGetNumFields(p)-1]
 * Returns: a pointer to the internal array.
 * Comments: if the internal array has not been allocated yet,
 *   this call causes it to be allocated.
 */
unsigned char *GRDValueIGetFieldValid(GRDValueI *p, mi_integer position);

/*
 * Name: GRDValueIGetDimName
 * Purpose: fetches the name of a particular dimension
 * Arguments:
 *    p: the GRDValueI
 *    dimNum: the index of the field to be given a name
 * Returns: the dim name as a text string, or NULL if no name has been
 *    assigned.
 * Comments: the text returned is property of the GRDValueI. Don't modify
 *    or free it directly.
 */
char *GRDValueIGetDimName(GRDValueI *p, int dimNum);

/*
 * Name: GRDValueIGetFieldName
 * Purpose: fetches the name of a particular field
 * Arguments:
 *    p: the GRDValueI
 *    fieldNum: the index of the field to be given a name
 * Returns: the field name as a text string, or NULL if no name has been
 *    assigned.
 * Comments: the text returned is property of the GRDValueI. Don't modify
 *    or free it directly.
 */
char *GRDValueIGetFieldName(GRDValueI *p, int fieldNum);

/*
 * Name: GRDValueIDoesWrap
 * Purpose: determines is coordinate wrapping is enabled.
 * Arguments:
 *   p: the GRDValueI
 *   dim: the dimension
```

```
 * Returns: true if the grid wraps in the specified dimension
 */
int GRDValueIDoesWrap(GRDValueI *p, int dim);
```

Converting between a GRDValueI and a GRDValue (var binary) Type

In an ESQL/C application, the GRDValue is represented by the Informix var binary type.

```
/*
 * Name: GRDValueIToVarBinary
 * Purpose: Converts a GRDValueI to a GRDValue (inside a var binary type)
 *  so that it can be used in ESQL/C calls.
 * Arguments:
 *  p: the GRDValueI
 * Returns:
 *   the new GRDValue as a var binary.
 */
var binary GRDValueIToVarBinary(GRDValueI *p);

/*
 * Name: VarBinaryToGRDValueI
 * Purpose: Converts a GRDValue (inside a var binary type) to a GRDValueI
 * so that it can be used in ESQL/C calls.
 * Arguments:
 *  p: the GRDValue as a var binary.
 * Returns:
 *   the new GRDValueI.
 */
GRDValueI *VarBinaryToGRDValueI( var binary p);
```

Creating and Setting a GRDSpecI Type

```
/*
 * Name: GRDValueINew
 * Purpose: Allocate a new GRDValueI structure
 * Returns: the new structure.
 */
GRDSpecI *GRDSpecINew();

/*
 * Name: GRDSpecISetDims
 * Purpose: Establishes the number of dimensions held by the GRDValueI
 * Arguments:
 *    p: the GRDValueI
```

```
 *    numDims: number of dimensions
 *    numSamples: number of samples in each dimension
 *    axisIds: name of each dimensional axis
*/
void GRDSpecISetDims(GRDSpecI *p, mi_integer numDims, mi_integer
numSamples[], mi_integer axisIds[]);

/*
 * Name: GRDSpecISetAligned
 * Purpose: sets the grid basis vectors for dimensions for the simple
 *    case that the ith dimension is aligned along the ith
 *    axis.
 * Arguments:
 *    p: the GRDSpecI
 *    stepValues: the step value along each axes of the native
 *        coordinate system.
 * Comments: this is an alternative to using GRDValueISetIthDim.
 */
void GRDSpecISetAligned(GRDSpecI *p, mi_double_precision stepValues[]);

/*
 * Name: GRDSpecISetNonUniform
 * Purpose: define a nonuniformly sampled dimension
 * Arguments:
 *   p: the GRDSpecI
 *   dimNum: the number of the dimension that should be nonuniform
 *   samplePoints: where to sample along the dimNumth dimension.
 *      These values represent scaling factors applied to the dimensions
 *      stepping vector.
 */
void GRDSpecISetNonUniform(GRDSpecI *p, mi_integer dimNum,
mi_double_precision samplePoints[]);

/*
 * Name: GRDSpecISetNumFields
 * Purpose: sets the number of data fields that can be stored.
 * Arguments:
 *   p: the GRDSpecI
 *   numFields: the number of fields p should have.
 */
void GRDSpecISetNumFields(GRDSpecI *p, mi_integer numFields);

/*
 * Name: GRDSpecISetFieldType
 * Purpose: sets the parameters describing one of the fields
 * Arguments:
```

```
 *   p: the GRDSpecI
 *   fieldNum: which field to set (a value between 0 and numDataFields -1)
 *   fieldId: an integer id to be assigned to the field
 *   fieldType: the datatype of the field's elements (e.g., grdFieldREAL4).
 */
void GRDSpecISetFieldType(GRDSpecI *p, mi_integer fieldNum, mi_integer fieldId,
GRDFieldType fieldType);

/*
 * Name: GRDSpecIAppendFieldType
 * Purpose: sets the parameters describing the next available field. Returns
 *   the index of the field.
 * Arguments:
 *   p: the GRDSpecI
*   fieldId: an integer id to be assigned to the field
 *   fieldType: the datatype of the field's elements (e.g., grdFieldREAL4).
 * Comments: this is an alternative to using GRDSpecISetNumFields and
 *   GRDSpecISetFieldType, useful when you do not know the number of fields
 *   in advance and consecutive ordering of the fields is sufficient.
 */
mi_integer GRDSpecIAppendField(GRDSpecI *p, mi_integer fieldId, GRDFieldType
fieldType);

/*
 * Name: GRDSpecISetInterpolation
 * Purpose: sets the interpolation flag for a particular dimension.
 * Arguments:
 *   p: the spec
 *   dim: which dimensions (0..grdMaxDIMENSIONS)
 *   value: GRDInterpNEAREST or GRDInterpBILINEAR
 * Comments: the default is GRDInterpNEAREST.
 */
void GRDSpecISetInterpolation(GRDSpecI *p, mi_integer dim, mi_integer value);

/*
 * Name: GRDSpecIAddFieldName
 * Purpose: sets the textual name of a field
 * Arguments:
 *    p: the GRDSpecI
 *    fieldName: the textual name.
 * Comments: a copy is made of fieldName, the parameters doesn't
 *    need to be preserved for the length of the object.
 */
void GRDSpecIAddFieldName(GRDSpecI *p, char *fieldName);

/*
```

```
 * Name: GRDSpecISetDimName
 * Purpose: sets the textual name of a field
 * Arguments:
 *    p: the GRDSpecI
 *    dimNum: the index of the field to be given a name
 *    dimName: the textual name.
 * Comments: a copy is made of dimName, the parameters doesn't
 *    need to be preserved for the length of the object.
 */
void GRDSpecISetDimName(GRDSpecI *p, int dimNum, char *dimName);

/*
 * Name: GRDSpecIDoesWrap
 * Purpose: determines if the coordinate system wraps in a particular dimension
 * Arguments:
 *    p: the spec
 *    dimNum: the number of the dimension [0..grdMaxDIMENSIONS)
 * Returns:
 *    true if the dimensions wraps.
 */
int GRDSpecIDoesWrap(GRDSpecI *p, int dimNum);

/*
 * Name: GRDSpecISetSrText
 * Purpose: Sets the srtext of the spec.
 * Arguments:
 *    p: the spec
 *    srtext: the spatial reference text.
 */
void GRDSpecISetSrText(GRDSpecI *p, char *srtext);

/*
 * Name: GRDSpecISetWrap
 * Purpose: sets coordinate wrapping for a particular dimension.
 * Arguments:
 *   p: the spec
 *   dimNum: the dimension
 *   minVal: low end of the range
 *   maxVal: high end of the range
 */
void GRSpecISetWrap(GRDSpecI *p, int dimNum, double minVal, double maxVal);
```

Converting between a GRDSpecI and a GRDSpec (var binary) Type

The GRDSpecI is represented on a client by a var binary.

```
/*
 * Name: GRDSpecIToVarBinary
 * Purpose: Converts a GRDSpecI to a GRDSpec inside in a var binary.
 * Arguments:
 *   p: the GRDSpecI
 * Returns the var binary.
 */
var binary GRDSpecIToVarBinary(GRDSpecI *p);
```

Example: Inserting Data Using ESQL/C

The following ESQL/C program inserts a small grid into the gridtesttable in a database called gridtest. It makes reference to some error checking procedures that are ESQL/C specific and have nothing to do with the Grid DataBlade. These procedures can be found in the demoCode/esqlc directory along with the program below.

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <grd.h>
#include <grdError.h>
#include <grdValueI.h>
#include <grdStorage.h>
#include <grdTransfer.h>
#include <mi.h>

EXEC SQL include sqltypes;
EXEC SQL include exp_chk.ec;

GRDValueI *BuildGRDValue()
{
   static double startPt[] = { 0, 0, 0, 0 };
   static mi_integer axisIds[] = { 0, 1, 2, 0 };
   static mi_integer numSamples[] = { 4, 8, 8, 0};
   int i, j, k;
   static mi_integer tileSizes[grdMaxDIMENSIONS] = { 2, 4, 4, 1};
   int *dataP;
   GRDValueI *theValue;
   theValue = GRDValueINew();

   GRDValueISetDims(theValue,  sizeof(startPt)/sizeof(startPt[0]),
                    numSamples, axisIds);
   GRDValueISetStartPt(theValue, startPt);
   GRDValueISetFinalDimensions(theValue, numSamples);
   GRDValueISetSrid(theValue, 0);
   GRDValueISetTileSize(theValue, tileSizes);
   GRDValueIDisableMissing(theValue);

   GRDValueIAppendField(theValue, 5, grdFieldINT4);
   dataP = (int *)GRDValueIGetFieldData(theValue, 0);
   for( i = 0; i < numSamples[0]; i++ ) {
```

```
      for( j = 0; j < numSamples[1]; j++ ) {
         for( k = 0; k < numSamples[2]; k++ ) {
            for( m = 0; m < numSamples[3]; m++ ) {
               *dataP = i * 100*100 + j * 100 + k+1;
               dataP++;
            }
         }
      }
   }
   return theValue;
}


void LoadGridToTable(GRDValueI* p)
{
   EXEC SQL BEGIN DECLARE SECTION;
   var binary "GRDValue" gridVar;
   EXEC SQL END DECLARE SECTION;
   EXEC SQL whenever sqlerror CALL ignore206;
   EXEC SQL connect to 'gridtest' ;
   gridVar = GRDValueIToVarBinary(p);
   EXEC SQL delete from gridTestTable;
   EXEC SQL insert into gridtesttable values(:gridVar);
   EXEC SQL disconnect current;
}


int main()
{
   GRDValueI *theValue;
   theValue = BuildGRDValue();
   LoadGridToTable(theValue);
   exit(0);
}
```

Example: Extracting Data Using ESQL/C

The following ESQL/C program extracts a small grid from gridtesttable in a database called gridtest. It makes reference to some error checking procedures that are ESQL/C specific and have nothing to do with the Grid DataBlade. These procedures can be found in the demoCode/esqlc directory along with the program below.

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <unistd.h>
#include <malloc.h>
#include <mi.h>
#include <grd.h>
#include <grdStorage.h>
#include <grdInterpSupport.h>
#include <grdValueI.h>
```

```
EXEC SQL include exp_chk.ec;

GRDValueI *
UnloadGridFromTable(GRDSpecI *specValue, int srid ) {
   GRDValueI *result;
   EXEC SQL BEGIN DECLARE SECTION;
      int outSrid;
      var binary "grdvalue" gridVar;
      var binary "grdspec" theSpec;
   EXEC SQL END DECLARE SECTION;
   EXEC SQL whenever sqlerror CALL ignore206;
   EXEC SQL connect to 'gridtest' ;

   outSrid = srid;
   theSpec = GRDSpecIToVarBinary(specValue);

   if( ifx_var_alloc(&gridVar, 10) ) {
      GRDUserError("ifx_var_alloc failed");
   }
   if( ifx_var_flag(&gridVar, 1)) {
      GRDUserError("ifx_var_flag failed");
   }

   EXEC SQL select grdextract2pass(a, :theSpec)
      into :gridVar from gridtesttable;

   result = GRDValueIFromVarBinary( gridVar );
   ifx_var_dealloc(gridVar);
   ifx_var_dealloc(theSpec);

   EXEC SQL disconnect current;
   return result;
}


/*
 * query user for extraction values
 */
GRDSpecI *BuildGRDSpec()
{
   GRDSpecI *theSpec;
   static double startPt[] = { 0,0,0,0};
   static mi_integer sampleSize[] = { 1, 1, 5, 5};

   theSpec = GRDSpecINew();
   GRDSpecISetNumFields(theSpec, 1);
   GRDSpecISetFieldId(theSpec, 0, 5);

   GRDSpecISetDims(theSpec, 4, sampleSize, NULL);
   GRDSpecISetStartPt(theSpec, startPt);
   return theSpec;
}

static void DumpGrid(GRDValueI *extract)
{
   int i, j, k, m;
```

```
   int numDims;
   mi_integer *numSamples;
   int *data;

   numDims = GRDValueIGetNumDims(extract);
   numSamples = GRDValueIGetDimensions(extract);
   data = GRDValueIGetFieldData(extract, 0);
   for( i = 0; i < numSamples[0]; i++ ) {
      printf("==section %d===\n", i);
      for( j = 0; j < numSamples[1]; j++ ) {
         for( k = 0; k < numSamples[2]; k++ ) {
            for( m = 0; m < numSamples[3]; m++ ) {
              printf("%9d", *data);
              data++;
            }
            printf("\n");
         }
      }
   }
}

int main()
{
   GRDSpecI *theSpec;
   GRDValueI *extract;
   int outSrid = -1;
   GRDErrSource(argv[0]);
   theSpec = BuildGRDSpec();
   extract = UnloadGridFromTable(theSpec, outSrid);
   DumpGrid(extract);
   exit(0);
}
```

Java Client API

Class Descriptions

The Java client API is a set of classes that are used by Java client applications to store and extract grids from the Informix Server. They are found in the package *com.barrodale.grid* supplied with the Grid DataBlade. The Java API may be used in conjunction with the SQL API and in place of or in conjunction with the C API.

The classes fall under four categories:

- **GRDType Mirrors**: The GRDValue, and GRDSpec Java classes mirror the SQL Types of the same name. AbstractValue is a superclass that embodies the common aspects of GRDValue and GRDSpec.
- **Element Arrays**: Int1Field, Int2Field, Int4Field, Real4Field, Real8Field and Uint1Field implement multi-dimensional arrays of different primitive types. AbstractDataField is the superclass for these.
- **Support Classes**: GRDSqlIn, GRDSqlOut and GRDSupport provide support for the other classes. These classes are never used by applications directly.
- **Exceptions**: Six additional exception classes are defined by the Grid DataBlade. These are unchecked exceptions and are thrown when a programmer error is encountered. The exceptions are: BadArgumentException, BadOrderException, IncompatibleDimensionsException, InconsistentArityException, NullArgumentException, and UnknownArityException. The usual JDBC exceptions are also thrown when an SQL is encountered.

More detail about these classes can be found in the associated Javadocs.

Example: Inserting Data Using Java

The following sample class inserts a 2x3x2x1 grid of ascending byte values to field index 99 of the myField field of a table called myTable. It also includes driver-loading code.

```
import com.informix.jdbc.*;
import com.barrodale.grid.*;
import java.sql.*;
import java.io.*;
import java.lang.*;

public class TestInsert  {

public static void main(String[] arg)
{

try
{
```

```
  // load driver and connect to the database
  Driver IfmxDrv = (Driver)
      Class.forName("com.informix.jdbc.IfxDriver").newInstance();
  String url = "jjdbc:informix-sqli://myhost:myport/mydatabase:"  +
      "informixserver=ifmxdba;user=myusername;password=mypass";
  Connection con = DriverManager.getConnection(url);

  // registering classes with the type maps
  java.util.Map customtypemap = con.getTypeMap();
  if (customtypemap == null) { System.exit(-1); }
  customtypemap.put("grdvalue",
      Class.forName("com.barrodale.grid.GRDValue"));
  customtypemap.put("grdspec",
      Class.forName("com.barrodale.grid.GRDSpec"));

  // create the 2x5 data array
  byte[] dataArray = new byte[] {0,1,2,3,4,5,6,7,8,9,10,11};
  int xSize=2; int ySize=3; int zSize=2, tSize = 1;

  // create a GRDValue object containing the data
  int[] fieldDim = {xSize, ySize, zSize, tSize};
  Int1Field dataField = new Int1Field(99, fieldDim);

  int x=0; int y=0; int z=0; int z = 0int i=0;
  while (i < xSize*ySize*zSize*tSize)
  {
    byte dataPiece = dataArray[i++];
    int[] pos = new int[] {x, y, z};
    dataField.setElement(pos, dataPiece);
    x++;
    if (x >= xSize) { x = 0;  y++; }
    if (y >= ySize) { y = 0;  z++; }
    if (z >= zSize) { z = 0;  t++; }
  }

  GRDValue gValue = new GRDValue();
  gValue.setNumSamples(fieldDim);
  gValue.addField(dataField);
  gValue.setStartPt( new double [] { 0, 0, 0, 0 });

  // insert the GRDValue object into the database
  String pstmtSQL = "INSERT INTO myTable (myField) VALUES(?)";
  PreparedStatement pstmt = con.prepareStatement(pstmtSQL);
  pstmt.setObject (1, gValue, java.sql.Types.LONGVARBINARY);
  pstmt.executeUpdate();

}
catch (Exception e)
{
  System.out.println("Error: " + e.getMessage());
  e.printStackTrace();
  System.exit(-1);
}
System.out.println("Insert Successful");

}
}
```

Example: Extracting Data Using Java

The following sample class extracts the entire grid of data that was inserted in the example above. The dimensions are set to {2,3,2,1} and start point is set to {0,0,0,0}. The type of interpolation used is linear interpolation. This example includes loading the driver, connecting to the database and registering three classes with the type map.

```
import com.informix.jdbc.*;
import com.barrodale.grid.*;
import java.sql.*;
import java.io.*;
import java.lang.*;

public class TestExtract  {

public static void main(String[] arg)
{
try
{

  // load the driver and connect to the server
  Driver IfmxDrv = (Driver)
      Class.forName("com.informix.jdbc.IfxDriver").newInstance();
  String url = "jjdbc:informix-sqli://myhost:myport/mydatabase:"  +
      "informixserver=ifmxdba;user=myusername;password=mypass";
  Connection con = DriverManager.getConnection(url);

  // register classes with the type maps
  java.util.Map customtypemap = con.getTypeMap();
  if (customtypemap == null) System.exit(-1);
  customtypemap.put("grdvalue",
      Class.forName("com.barrodale.grid.GRDValue"));
  customtypemap.put("grdspec",
      Class.forName("com.barrodale.grid.GRDSpec"));

  // extract the GRDValue object from the database
  int xSize=2; int ySize=3; int zSize=2, tSize = 1;
  GRDSpec gSpec = new GRDSpec();
  gSpec.setNumSamples(new int[] {xSize, ySize, zSize, tSize});
  gSpec.setFieldIds(new int[] {99});
  gSpec.setIdentityBasis();
  gSpec.setStartPt(new double[] {0,0,0,0});

  String pstmtSQL = "SELECT GRDExtract( myField, ?) from myTable";
  PreparedStatement pstmt = con.prepareStatement(pstmtSQL);
  pstmt.setObject(1, gSpec);
  ResultSet rs = pstmt.executeQuery();

  rs.next();
  GRDValue gValue = (GRDValue)rs.getObject(1);
  Int1Field dataField = (Int1Field)gValue.getFieldByPosition(0);
  byte[] dataArray = (byte[])dataField.getElements();
```

```
  // display array contents
  int i=0;
  while (i < dataArray.length && i < 100)
  {
    System.out.print( dataArray[i]+", ");
    if (i%10 == 9) System.out.println();
    i++;
  }

}
catch (Exception e)
{
  System.out.println("\nError: " + e.getMessage());
  e.printStackTrace();
  System.exit(-1);
}
System.out.println("\nExtract Successful");

}
}
```

Appendix A: Examples of Grid-Spatial Conversion

In the following examples, the data stored in the 2-D source grid is the sequence 1, 2, 3, ... , 28. The grid dimensions are 4x7x1x1 (4 in the *x* dimension, 7 in the *y* dimension) as shown below:

7	14	21	28
6	13	20	27
5	12	19	26
4	11	18	25
3	10	17	24
2	9	16	23
1	8	15	22

Note that the first element is at the bottom left and that grid positions increase vertically, to be consistent with spatial coordinates, which are usually represented this way.

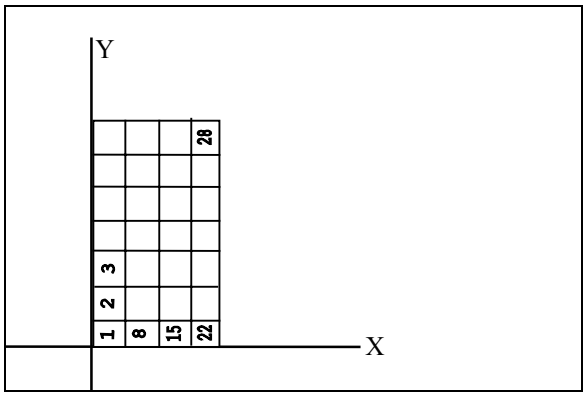
Default Grid Meta-Data

The default grid meta-data is as follows:

STARTPOINT: 0 0 0 0
BASIS 0: 1 0 0 0
BASIS 1: 0 1 0 0
BASIS 2: 0 0 1 0
BASIS 3: 0 0 0 1

The interpretation of this grid is:

a value of 1 at location (0, 0, 0, 0)
a value of 2 at location (0, 1, 0, 0)
a value of 3 at location (0, 2, 0, 0)
...
a value of 8 at location (1, 0, 0, 0)
...
a value of 15 at location (2, 0, 0, 0)
...
a value of 28 at location (3, 6, 0, 0)



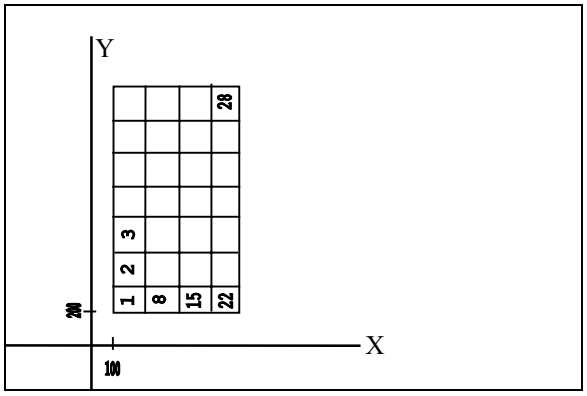
Translated Grid

By setting the start point, we translate the spatial coordinates of the grid points. The following meta-data offsets the *x* coordinates by 100 and the *y* coordinates by 200:

STARTPOINT: 100 200 0 0
BASIS 0: 1 0 0 0
BASIS 1: 0 1 0 0
BASIS 2: 0 0 1 0
BASIS 3: 0 0 0 1

The interpretation of this grid is:

a value of 1 at location (100, 200, 0, 0)
a value of 2 at location (100, 201, 0, 0)
a value of 3 at location (100, 202, 0, 0)
...
a value of 8 at location (101, 200, 0, 0)
...
a value of 15 at location (102, 200, 0, 0)
...
a value of 28 at location (103, 206, 0, 0)



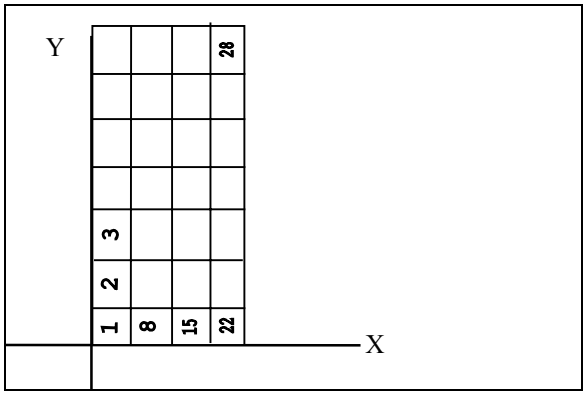
Scaled Grid

A scaling of 1.2 along the *x* coordinates (the first dimension) and 3.5 along the *y* coordinates can be done by setting the diagonals of the BASIS vector 1.2 and 3.5, respectively. The new version is:

STARTPOINT: 0 0 0 0
BASIS 0: 1.2 0 0 0
BASIS 1: 0 3.5 0 0
BASIS 2: 0 0 1 0
BASIS 3: 0 0 0 1

The interpretation of this grid is:

a value of 1 at location (0, 0, 0, 0)
a value of 2 at location (0, 3.5, 0, 0)
a value of 3 at location (0, 7, 0, 0)
...
a value of 8 at location (1.2, 0, 0, 0)
...
a value of 15 at location (2.4, 0, 0, 0)
...
a value of 28 at location (3.6, 21, 0, 0)



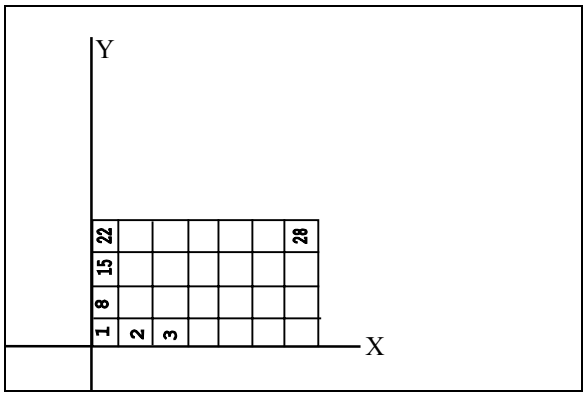
Column Major Scan Order

The *x* and *y* coordinates can be flipped by swapping the basis vectors. The new version is:

STARTPOINT: 0 0 0 0
BASIS 0: 0 1.2 0 0
BASIS 1: 3.5 0 0
BASIS 2: 0 0 1 0
BASIS 3: 0 0 0 1

The interpretation of this grid is:

a value of 1 at location (0, 0, 0, 0)
a value of 2 at location (3.5, 0, 0, 0)
a value of 3 at location (7, 0, 0, 0)
...
a value of 8 at location (0, 1.2, 0, 0)
...
a value of 15 at location (0, 2.4, 0, 0)
...
a value of 28 at location (21, 3.6, 0, 0)



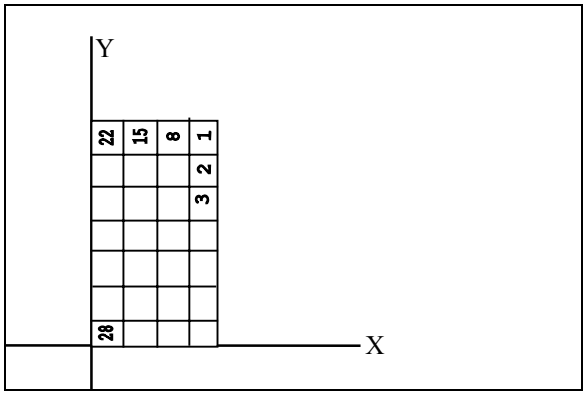
Scan Direction

By adjusting the STARTPOINT and the direction of the BASIS vectors, different scan directions can be achieved.

STARTPOINT: 3 6 0 0
BASIS 0: –1 0 0 0
BASIS 1: 0 –1 0 0
BASIS 2: 0 0 1 0
BASIS 3: 0 0 0 1

The interpretation of this grid is:

a value of 1 at location (3, 6, 0, 0)
a value of 2 at location (3, 5, 0, 0)
a value of 3 at location (3, 4, 0, 0)
...
a value of 8 at location (2, 6, 0, 0)
...
a value of 15 at location (1, 6, 0, 0)
...
a value of 28 at location (0, 0, 0, 0)



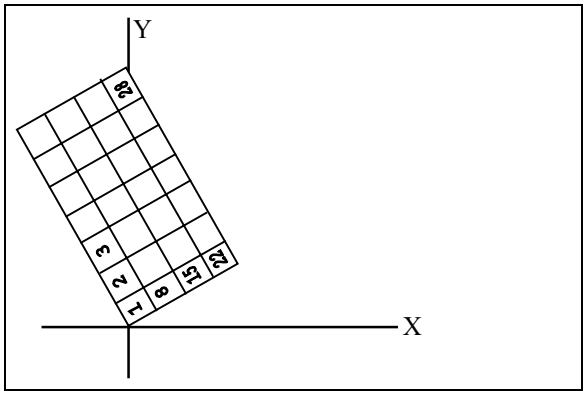
Rotation

Rotation can be done using setting the BASIS vectors to the appropriate sine and cosine values. The example below rotates 30 degrees from the positive side of dimension 0, to the positive side of dimension 1 (a counterclockwise rotation if dimension 0 is mapped to *x*, and dimension 1 is mapped to *y*, and plotted).

STARTPOINT: 0 0 0 0
BASIS 0: 0.866 –0.5 0 0
BASIS 1: 0.5 0.866 0 0
BASIS 2: 0 0 1 0
BASIS 3: 0 0 0 1

The interpretation of this grid is:

a value of 1 at location (0, 0, 0, 0)
a value of 2 at location (–0.5, 0.866, 0, 0)
a value of 3 at location (–1.0, 1.732, 0, 0)
...
a value of 8 at location (0.866, 0.5, 0, 0)
...
a value of 15 at location (1.732, 1.0, 0, 0)
...
a value of 28 at location (–0.402, 6.696, 0, 0)



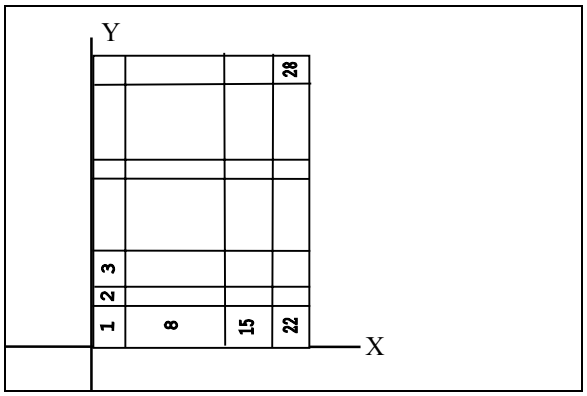
Non-Uniform Grid Spacing

The key point of non-uniform grid spacing is that (in the absence of distorting BASIS vectors) the resulting region remains rectilinear since each axis is stretched independently.

STARTPOINT: 0 0 0 0
BASIS 0: 1 0 0 0
BASIS 1: 0 1 0 0
BASIS 2: 0 0 1 0
BASIS 3: 0 0 0 1
NONUNIFORM 0: 0 1 5 10
NONUNIFORM 1: 0 3 4 5 9 10 12

The interpretation of this grid is:

a value of 1 at location (0, 0, 0, 0)
a value of 2 at location (0, 3, 0, 0)
a value of 3 at location (0, 4, 0, 0)
...
a value of 8 at location (1, 0, 0, 0)
...
a value of 15 at location (5, 0, 0, 0)
...
a value of 28 at location (10, 12, 0, 0)

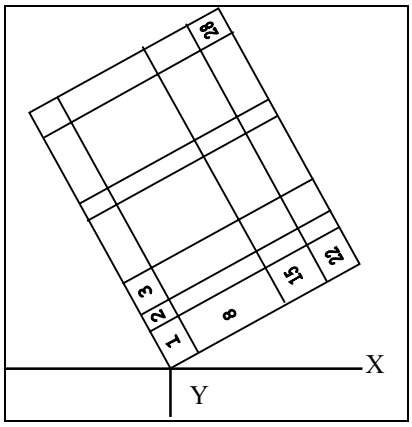


Combining Non-Uniform Grid Spacing with Rotation

The key to combining non-uniform grid spacing with BASIS vector transformations (other than the identity) is that non-uniform spacing is processed before the BASIS vector transformations. Among other effects, this allows data be extracted at arbitrary points along a line. The following meta-data combines the earlier rotation with the above grid spacing.

STARTPOINT: 0 0 0 0
BASIS 0: 0.866 –0.5 0 0
BASIS 1: 0.5 0.866 0 0
BASIS 2: 0 0 1 0
BASIS 3: 0 0 0 1
NONUNIFORM 0: 0 1 5 10
NONUNIFORM 1: 0 3 4 5 9 10 12

The interpretation of this grid is:
a value of 1 at location (0, 0, 0, 0)
a value of 2 at location (–1.5, 2.598, 0, 0)
a value of 3 at location (–2.0, 3.464, 0, 0)
...
a value of 8 at location (0.866, 0.5, 0, 0)
...
a value of 15 at location (4.33, 2.5, 0, 0)
...
a value of 28 at location (2.66, 15.392, 0, 0)



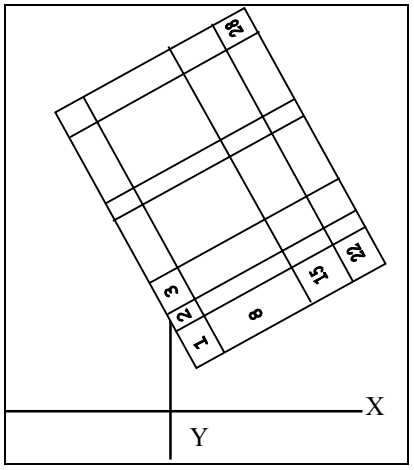
Combining Non-Uniform Grid Spacing with Rotation and Translation

The key point in combining other effects with translation is to remember that the translation is the last step in the processing.

STARTPOINT: 100 200 0 0
BASIS 0: 0.866 –0.5 0 0
BASIS 1: 0.5 0.866 0 0
BASIS 2: 0 0 1 0
BASIS 3: 0 0 0 1
NONUNIFORM 0: 0 1 5 10
NONUNIFORM 1: 0 3 4 5 9 10 12

The interpretation of this grid is:

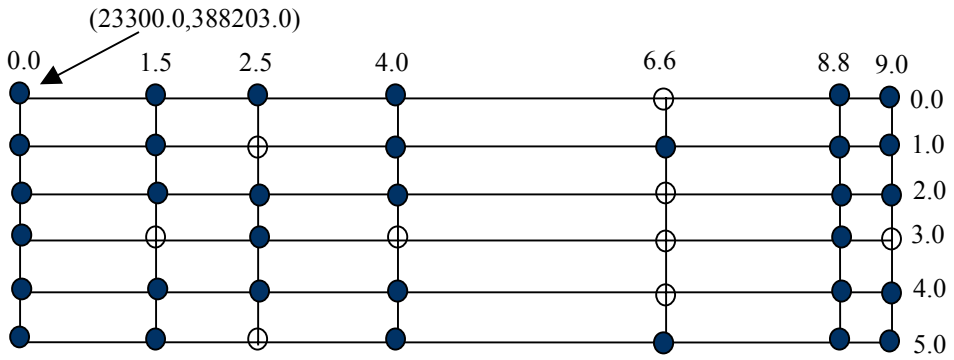
a value of 1 at location (100, 200, 0, 0)
a value of 2 at location (98.5, 202.598, 0, 0)
a value of 3 at location (98.0, 203.464, 0, 0)
...
a value of 8 at location (100.866, 200.5, 0, 0)
...
a value of 15 at location (104.33, 202.5, 0, 0)
...
a value of 28 at location (102.66, 215.392, 0, 0)



Appendix B: Examples of GRDValue and GRDSpec

Original Grid Base Location and Sampling Locations

In the 2D grid diagram below, filled circles represent sampling points where there was data for the WindDirId field, while hollow circles represent sampling points where there was no data available. The values at the points in the diagram indicate the displacements relative to the start point (23300, 388203).



Values in a Field with id = WindDirId

8	7	3	8		3	8
3	4		3	4	5	3
7	4	55	7		55	7
4		48			48	
6	7	2	4		2	5
3	1		4	4	5	3

Values in a Field with id = WinSpeedId

1.8	7.3	3.5	1.8	7.5		1.8
3.3	1.4		3.3	1.4	0.5	3.3
7.4	4.4	5.1	7.4	4.4	5.1	7.0
4.1		4.8	4.1		4.8	
1.8	7.3	3.5	1.8	7.5		1.8
3.3	1.4		3.3	1.4	0.5	3.3

GRDValue Representation

SRID = 1

numFields = 2
startPt = { 23300.0, 388203.0, 0, 0 }
gridBasisVectors = {{1,0,0,0}, {0,1,0,0}, {0,0,1,0}, {0,0,0,1}}
fieldIds = { WindDirID, WindSpeedID}
fieldTypes integer = { grdFieldINT, grdFieldFLOAT}
dimensions = {7, 6, 1, 1 }
nonUniformSteps = {{0,1.5,2.5,4,6.6,8.8,9}, {},{}, {}}
tileSizes = {3, 4, 1, 1}
tiles = {

```
 {                                                          // tile 0
      {{ 8,3,7,4}, {7,4,4,_}, {3,_,55,48}}                  // WinDirID
      {{ t,t,t,t}, {t,t,t,f}, {t,f,t,t}}                    // WinDirID flags
      {{ 1.8,3.3,7.4,4.1}, {7.3,1.4,4.4,_}, {3.5,_,5.1,4.8}}  // WinSpeedId
      {{ t,t,t,t}, {t,t,t,f}, {t,f,t,t}}                    // WinSpeedID flags
 }
 {                                                          // tile 1
      {{ 6,3,_,_}, {7,1,_,_}, {2,_,_,_}}                    // WinDirID
      {{ t,t,f,f}, {t,t,f,f}, {t,f,f,f}}                    // WinDirID flags
      {{ 1.8,3.3,_,_}, {7.3,1.4,_,_}, {3.5,_,_,_}}          // WinSpeedId
      {{ t,t,f,f}, {t,t,f,f}, {t,f,f,f}}                    // WinSpeedID flags
 }
 {                                                          // tile 2
      {{ 8,3,7,_}, {_,4,_,_}, {3,5,55,48}}                  // WinDirID
      {{ t,t,t,f}, {f,t,f,f}, {t,t,t,t}}                    // WinDirID flags
      {{ 1.8,3.3,7.4,4.1}, {7.5,1.4,4.4,_}, {_,0.5,5.1,4.8}}  // WinSpeedId
      {{ t,t,t,t}, {t,t,t,f}, {f,t,t,t}}                    // WinSpeedID flags
 }
 {                                                          // tile 3
      {{ 4,4,_,_}, {_,4,_,_}, {2,5,_,_}}                    // WinDirID
      {{ t,t,f,f}, {f,t,f,f}, {t,t,f,f}}                    // WinDirID flags
      {{ 1.8,3.3,_,_}, {7.5,1.4,_,_}, {_,0.5,_,_}}          // WinSpeedId
      {{ t,t,f,f}, {t,t,f,f}, {f,t,f,f}}                    // WinSpeedID flags
 }
 {                                                          // tile 4
      {{ 8,3,7,_}, {_,_,_,_}, {_,_,_,_}}                    // WinDirID
      {{ t,t,t,f}, {f,f,f,f}, {f,f,f,f}}                    // WinDirID flags
      {{ 1.8,3.3,7.0,_}, {_,_,_,_}, {_,_,_,_}}              // WinSpeedId
      {{ t,t,t,f}, {f,f,f,f}, {f,f,f,f}}                    // WinSpeedID flags
 }
 {                                                          // tile 5
      {{ 5,3,_,_}, {_,_,_,_}, {_,_,_,_}}                    // WinDirID
      {{ t,t,f,f}, {f,f,f,f}, {f,f,f,f}}                    // WinDirID flags
      {{ 1.8,3,3_,_}, {_,_,_,_}, {_,_,_,_}}                 // WinSpeedId
      {{ t,t,f,f}, {f,f,f,f}, {f,f,f,f}}                    // WinSpeedID flags
 }
}
```

GRDSpec Representation

startPt = {0,0,0,0} // start point set to 0,0,0,0
gridBasisVectors = {} // grid step vector not set
nonUniformSteps = {{},{},{},{}} // no nonuniform steps
fieldIds = {} // field ids not set
dimensions = {7, 6, 1, 1} // the dimensions of the grid

Appendix C: Grid Import-Export Format (GIEF)

GIEF is the import/export format used by the Grid DataBlade. External conversion programs are employed to convert grid files between other formats (e.g., GeoTiff, GRIB, and dialects of NetCDF) and one or more GIEF files.

GIEF is a NetCDF dialect that is general enough to represent any grids of primitive scalar data elements (i.e., 1, 2 or 4-byte integers, 4 or 8-byte floating point values). It allows source grid files of various formats to have one internal uniform representation.

A key feature of GIEF is that there is a simple mapping from GIEF files to database tables. This mapping ensures that any database programmer/DBA will be able to verify that GIEF file importing and exporting is error-free, simply by inspecting the database contents.

Features

- Industry-standard mapping projections
- Affine transformations (i.e., translation, scaling and rotation)
- 4D grids (with 2D and 3D grids as special cases)
- Global attributes (scalar or 1D vector[4]; e.g., text strings and valid ranges)
- Grid-specific variable attributes (scalar or 1D vector; e.g., a fill value)
- Optional mapping of integer values to floating point values.
- Nonuniform axes

Limitations

- All grids in a GIEF file must have the same dimensions.

[4] The first release of the Grid DataBlade with GIEF support will only handle scalar attributes and text strings. The ability to handle ranges or arrays will not be present until a later release.

- A maximum of ten grids may be stored in a single GIEF file.[5]

Note: A simple strategy for overcoming these limitations is for the conversion programs to generate multiple GIEF files from a single source file.

Conventions

Grid Size

- Every GIEF file has exactly four dimensions declared. To support 1D, 2D or 3D grids, one or more of these dimensions will have a value of 1.
- The dimensions must be listed in the same order in all grid variable declarations in a particular GIEF file.

Supporting Grids that Wrap

Grid dimensions that should wrap around (e.g., columns in a grid expressed in a geographic reference system that spans the range 0° – 360°), should be denoted by an attribute called grid_wraps_in. The grid_wraps_in attribute should have as a value a comma-separated list of dimension names that can wrap. For example, if the "column" dimension should wrap in a particular GIEF file, the GIEF file should contain the line:

:grid_wraps_in = "column";

Mapping Projection

Each GIEF file has a special global attribute called "srtext" whose value is the OGC well-known-text form of a spatial reference system. A converter that deals with only one type of projection can construct a suitable string by concatenating a few string literals with some integer and floating point values.

A formal description of spatial reference text can be found in http://www.opengis.org/techno/specs/99-049.rtf on page 70; however, the following example of the spatial reference text for Lambert Conformal projection is taken from http://www.opengis.org/techno/interop/EPSG2WKT.TXT.

PROJCS["Madrid 1870 (Madrid) / Spain",GEOGCS["Madrid 1870 (Madrid)",DATUM["Madrid_1870",SPHEROID["Struve 1860",6378298.3,294.73]],PRIMEM["Madrid",-3.68793888888889],UNIT["degree",0.0174532925199433]],PROJECTION["Lambert_C

[5] In the case of FNMOC data sets, typically a single grid will be stored in a single file, with the exception being vector data (e.g., wind components).

onformal_Conic_1SP"],PARAMETER["latitude_of_origin",40],PARAMETER["central_meridian",0],PARAMETER["scale_factor",0.9988085293],PARAMETER["false_easting",600000],PARAMETER["false_northing",600000],UNIT["metre",1]]

All coordinate systems supported by the IBM/Informix Spatial DataBlade can be used by GIEF. In addition to geographic coordinate systems, this includes the following planar coordinate systems:

Aitoff, Albers, Azimuthal_Equidistant,
Behrmann, Bonne, Cassini, Craster_Parabolic, Cylindrical_Equal_Area,
Double_Stereographic, Eckert_I, Eckert_II, Eckert_III, Eckert_IV, Eckert_V, Eckert_VI,
Equidistant_Conic, Equidistant_Cylindrical, Flat_Polar_Quartic,
Gall_Stereographic, Gauss_Kruger, Gnomonic, Hammer_Aitoff,
Hotine_Oblique_Mercator_Azimuth_Center,
Hotine_Oblique_Mercator_Azimuth_Natural_Origin,
Hotine_Oblique_Mercator_Two_Point_Center,
Hotine_Oblique_Mercator_Two_Point_Natural_Origin,
Krovak, Lambert_Azimuthal_Equal_Area, Lambert_Conformal_Conic, Loximuthal,
Mercator, Miller_Cylindrical, Mollweide, New_Zealand_Map_Grid,
Orthographic, Plate_Carree, Polyconic, Quartic_Authalic, Robinson,
Sinusoidal, Stereographic, Times, Transverse_Mercator, Two_Point_Equidistant,
Van_der_Grinten_I, Vertical_Near_Side_Perspective,
Winkel_I, Winkel_II, Winkel_Tripel.

A list of sample spatial reference text for each of these projections (with the exception of Gauss_Kruger and Double_Stereographic) is given in Section "Sample Spatial Reference Text" below.

Translation

Each GIEF file must have a global attribute called "translation" which has four double precision values. The translation attribute denotes the location of the first grid element in the spatial projection.

To support projection transformations, the first and second of the four double precision values defining the translation must be the *x* (or longitude) and *y* (or latitude) values, respectively. The third value generally represents *z* (or equivalent), while the fourth value normally represents time.[6] (To support geodetic indexing, this convention must be followed.)

[6] A FNMOC requirement is that the fourth element of the translation value represents time and is always zero, to denote that times are expressed in absolute terms.

Affine Transformation

Each GIEF file may have an affine transformation expressed by a global attribute called "affine_transformation" in the form:

:affine_transformation = a1,a2,a3,a4,
b1,b2,b3,b4,
c1,c2,c3,c4,
d1,d2,d3,d4;

Grid locations will be mapped to spatial coordinates in the projection by the expressions:

spatial[1] = a1*gridloc[1] + a2*gridloc[2] + a3*gridloc[3] + a4*gridloc[4] + translation[1]
spatial[2] = b1*gridloc[1] + b2*gridloc[2] + b3*gridloc[3] + b4*gridloc[4] + translation[2]
spatial[3] = c1*gridloc[1] + c2*gridloc[2] + c3*gridloc[3] + c4*gridloc[4] + translation[3]
spatial[4] = d1*gridloc[1] + d2*gridloc[2] + d3*gridloc[3] + d4*gridloc[4] + translation[4]

Spatial[1] and spatial[2] will be treated as holding the *x* (or longitude) and *y* (or latitude) values, respectively, during any changes in projections.

If the affine_transformation attribute is not present, it will be assumed to have the value:

1,0,0,0,
0,1,0,0,
0,0,1,0,
0,0,0,1

which can be viewed as an identity matrix. The Grid DataBlade requires that the *affine_transformation* be invertible.

Nonuniform Axes

Nonuniform axes (i.e., those in which the values are not sampled at constant intervals along a particular axis) may be specified by 1D variables that are typed to a particular dimension. The name of the variable should be the same as the name as the dimension. The values are declared as single or double precision values and must be monotonically increasing. They effectively modify the gridloc[i] value in the equations described in the preceding section on affine transformations; i.e., the following statement:

double gridDim2(gridDim2);
gridDim2= 1,2,3,6,7,8,10,120,200;

results in the equations:

```
spatial[1] = a1*gridloc[1] + a2*gridDim2[gridloc[2]] + a3*gridloc[3] + a4*gridloc[4] + translation[1]
spatial[2] = b1*gridloc[1] + b2*gridDim2[gridloc[2]] + b3*gridloc[3] + b4*gridloc[4] + translation[2]
spatial[3] = c1*gridloc[1] + c2*gridDim2[gridloc[2]] + c3*gridloc[3] + c4*gridloc[4] + translation[3]
spatial[4] = d1*gridloc[1] + d2*gridDim2[gridloc[2]] + d3*gridloc[3] + d4*gridloc[4] + translation[4]
```

A more complete example is:

```
dimensions:
        time = 4;
        level = 3;
        row = 2;
        column = 2;
variables:
        float time(time);
        float level(level);
        float wind_u(time,level,row,column);
        ...
data:
        time = 1,4,10,11;
        level = 10,11,29;
        wind_u = 1.2,2.3,3.4,4.5,5.6,6.7,7.8,8.9,9.8,8.7,7.6,6.5,
                 5.4,4.3,3.2,2.1,1.2,2.3,3.4,4.5,5.6,6.7,7.8,8.9,
                 9.8,8.7,7.6,6.5,5.4,4.3,3.2,2.1,1.2,2.3,3.4,4.5,
                 5.6,6.7,7.8,8.9,9.8,8.7,7.6,6.5,5.4,4.3,3.2,2.1;
```

To support geodetic indexing, the time units must be in Unix Epoch time, notional seconds since the beginning of January 1, 1970, UTC. Some applications may require a non-uniform axis attribute for time.[7]

There may or may not be attributes for dimensions other than time since these can often be denoted by non-zero entries in a translation attribute.

Variable-Specific Attributes

The standard NetCDF *_FillValue* attribute is used to denote missing values. It is the only variable specific attribute which the Grid DataBlade will actively recognize. All other attributes are simply passed through to appropriately named columns in the database (see the section "Mapping Names from GIEF to the Database" below).

Other Attributes

There are a number of attributes (global or specific to variables) that GIEF does not interpret itself, but should be a part of GIEF files for the sake of converter programs that

[7] This is a FNMOC requirement.

need to convert GIEF to another format. These attributes are described in the document "Standard FNMOC GIEF Attributes".

Grid Variables

The only variables that may be present in the GIEF file are those described above (e.g., the nonuniform variables) and those holding the actual gridded data. All variables holding gridded data must have the same set of dimensions. The names of the grid variables are preserved in the database. Missing grid values are handled by NetCDF's *_FillValue* convention.

Mapping Names from GIEF to the Database

When a GIEF file is loaded into row of a database table, the information contained in its variables and attributes must be mapped to columns in the database. Database administrators need to know this mapping so they know what columns to include when defining a table. Database clients need to know this mapping so that they can search a table for rows based on particular attributes. Programmers writing converter programs must be aware of the name mappings to avoid producing names that inadvertently clash with each other. This mapping is as follows:

- 1D variables define the nonuniform axis characteristics of a grdvalue stored in a column called *grid*.
- *Srtext, translation*, and *affine_transformation* global attributes define the SRID and basis vector attributes of the grdvalue.
- Variables containing grids are represented as fields in the grdvalue. The names of the variables are stored in a column called *field_names* as a comma-separated list. For example, if the GIEF file contained the variables pressure, temperature, and salinity, the *field_names* column might have the value "pressure,temperature,salinity"' (the exact order of the names in the list depends on the order in which they appear in the GIEF file).
- The names of the dimensions are stored in a column called *dim_names* as a comma-separated list.
- A global attribute named $*t* is mapped to a database column called *g_*$*t*. For example, a global attribute called *center_id* would be mapped to a column called *g_center_id*.
- An attribute called $*t* of a variable called $*v* is mapped to a column called *l_*$*v*__$*t* (note that there are two underscores separating $*v* and $*t*). For example, an attribute called *range* of a variable called *depth* would be mapped to a column called *l_depth__range*.

An Example GIEF File (as a CDL File)

Note: this example does not include the FNMOC attributes described in the document "Standard FNMOC GIEF Attributes". A more complete GIEF file containing those attributes is provided in that document.

```
netcdf GIEF-2002031312-000-air_temp-NOGAPS {
dimensions:
        time = 1;
        level = 4 ;
        row = 4 ;
        column = 3 ;
variables:
        float air_temp(column,row,time,level) ;
        double level(level);
        double time(time);

// global attributes:
        :dtg = "2002031312" ;
        :model_name = "NOGAPS" ;
        :geom_name = "global_4x6" ;
        :lvl_type = "isbr_lvl" ;
        :unit_name = "K" ;
        :translation = 3000,3032,5,0;
        :affine_transformation = 1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1;
        :srtext =
"PROJCS['Madrid 1870 (Madrid) / Spain',GEOGCS['Madrid 1870 Madrid)',
DATUM['Madrid_1870',SPHEROID['Struve1860',6378298.3,294.73]],
PRIMEM['Madrid',-3.68793888888889],UNIT['degree',0.0174532925199433]],
PROJECTION['Lambert_Conformal_Conic_1SP'],
PARAMETER['latitude_of_origin',40],PARAMETER['central_meridian',0],
PARAMETER['scale_factor',0.9988085293],PARAMETER['false_easting',600000],
PARAMETER['false_northing',600000],UNIT['metre',1]]";

data:
 level = 0, 5, 6, 10;
 time = 1000300;
 air_temp =
   241.5278, 241.5278, 241.5278, 241.5278, 241.5278, 241.5278, 241.5278,
   241.5278, 241.5278, 241.5278, 241.5278, 241.5278, 241.5278, 241.5278,
   241.5278, 241.5278, 241.5278, 241.5278, 241.5278, 241.5278, 241.5278,
   241.5278, 241.5278, 241.5278, 241.5278, 241.5278, 241.5278, 241.5278,
   241.5278, 241.5278, 241.5278, 241.5278, 241.5278, 241.5278, 241.5278,
   241.5278, 241.5278, 241.5278, 241.5278, 241.5278, 241.5278, 241.5278,
   241.5278, 241.5278, 241.5278, 241.5278, 241.5278, 241.5278;
}
```

Sample Spatial Reference Text

The following projection code has been extracted from the IBM/Informix Spatial DataBlade using the *se_createsrtext()* function applied to selected factory ids.

```
Aitoff
        PROJCS["Sphere_Aitoff",GEOGCS["GCS_Sphere",DATUM["D_Sphere",
        SPHEROID["Sphere",6371000.0,0.0]],PRIMEM["Greenwich",0.0],UN
        IT["Degree",0.0174532925199433]],PROJECTION["Aitoff"],PARAME
        TER["False_Easting",0.0],PARAMETER["False_Northing",0.0],PAR
        AMETER["Central_Meridian",0.0],UNIT["Meter",1.0]]
Aitoff
        PROJCS["World_Aitoff",GEOGCS["GCS_WGS_1984",DATUM["D_WGS_198
        4",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM["Gre
        enwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTION["
        Aitoff"],PARAMETER["False_Easting",0.0],PARAMETER["False_Nor
        thing",0.0],PARAMETER["Central_Meridian",0.0],UNIT["Meter",1
        .0]]
Albers
        PROJCS["Africa_Albers_Equal_Area_Conic",GEOGCS["GCS_WGS_1984
        ",DATUM["D_WGS_1984",SPHEROID["WGS_1984",6378137.0,298.25722
        3563]],PRIMEM["Greenwich",0.0],UNIT["Degree",0.0174532925199
        433]],PROJECTION["Albers"],PARAMETER["False_Easting",0.0],PA
        RAMETER["False_Northing",0.0],PARAMETER["Central_Meridian",2
        5.0],PARAMETER["Standard_Parallel_1",20.0],PARAMETER["Standa
        rd_Parallel_2",23.0],PARAMETER["Latitude_Of_Origin",0.0],UNI
        T["Meter",1.0]]
Azimuthal_Equidistant
        PROJCS["Sphere_Azimuthal_Equidistant",GEOGCS["GCS_Sphere",DA
        TUM["D_Sphere",SPHEROID["Sphere",6371000.0,0.0]],PRIMEM["Gre
        enwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTION["
        Azimuthal_Equidistant"],PARAMETER["False_Easting",0.0],PARAM
        ETER["False_Northing",0.0],PARAMETER["Central_Meridian",0.0]
        ,PARAMETER["Latitude_Of_Origin",0.0],UNIT["Meter",1.0]]
Behrmann
        PROJCS["World_Behrmann",GEOGCS["GCS_WGS_1984",DATUM["D_WGS_1
        984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM["G
        reenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTION
        ["Behrmann"],PARAMETER["False_Easting",0.0],PARAMETER["False
        _Northing",0.0],PARAMETER["Central_Meridian",0.0],UNIT["Mete
        r",1.0]]
Bonne
        PROJCS["World_Bonne",GEOGCS["GCS_WGS_1984",DATUM["D_WGS_1984
        ",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM["Gree
        nwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTION["B
        onne"],PARAMETER["False_Easting",0.0],PARAMETER["False_North
        ing",0.0],PARAMETER["Central_Meridian",0.0],PARAMETER["Stand
        ard_Parallel_1",60.0],UNIT["Meter",1.0]]
```

```
Cassini
            PROJCS["World_Cassini",GEOGCS["GCS_WGS_1984",DATUM["D_WGS_19
            84",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM["Gr
            eenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTION[
            "Cassini"],PARAMETER["False_Easting",0.0],PARAMETER["False_N
            orthing",0.0],PARAMETER["Central_Meridian",0.0],PARAMETER["S
            cale_Factor",1.0],PARAMETER["Latitude_Of_Origin",0.0],UNIT["
            Meter",1.0]]

Craster_Parabolic
            PROJCS["World_Craster_Parabolic",GEOGCS["GCS_WGS_1984",DATUM
            ["D_WGS_1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],
            PRIMEM["Greenwich",0.0],UNIT["Degree",0.0174532925199433]],P
            ROJECTION["Craster_Parabolic"],PARAMETER["False_Easting",0.0
            ],PARAMETER["False_Northing",0.0],PARAMETER["Central_Meridia
            n",0.0],UNIT["Meter",1.0]]

Cylindrical_Equal_Area
            PROJCS["World_Cylindrical_Equal_Area",GEOGCS["GCS_WGS_1984",
            DATUM["D_WGS_1984",SPHEROID["WGS_1984",6378137.0,298.2572235
            63]],PRIMEM["Greenwich",0.0],UNIT["Degree",0.017453292519943
            3]],PROJECTION["Cylindrical_Equal_Area"],PARAMETER["False_Ea
            sting",0.0],PARAMETER["False_Northing",0.0],PARAMETER["Centr
            al_Meridian",0.0],PARAMETER["Standard_Parallel_1",0.0],UNIT[
            "Meter",1.0]]

Eckert_III
            PROJCS["World_Eckert_III",GEOGCS["GCS_WGS_1984",DATUM["D_WGS
            _1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM[
            "Greenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTI
            ON["Eckert_III"],PARAMETER["False_Easting",0.0],PARAMETER["F
            alse_Northing",0.0],PARAMETER["Central_Meridian",0.0],UNIT["
            Meter",1.0]]

Eckert_II
            PROJCS["World_Eckert_II",GEOGCS["GCS_WGS_1984",DATUM["D_WGS_
            1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM["
            Greenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTIO
            N["Eckert_II"],PARAMETER["False_Easting",0.0],PARAMETER["Fal
            se_Northing",0.0],PARAMETER["Central_Meridian",0.0],UNIT["Me
            ter",1.0]]

Eckert_I
            PROJCS["World_Eckert_I",GEOGCS["GCS_WGS_1984",DATUM["D_WGS_1
            984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM["G
            reenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTION
            ["Eckert_I"],PARAMETER["False_Easting",0.0],PARAMETER["False
            _Northing",0.0],PARAMETER["Central_Meridian",0.0],UNIT["Mete
            r",1.0]]

Eckert_IV
            PROJCS["World_Eckert_IV",GEOGCS["GCS_WGS_1984",DATUM["D_WGS_
            1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM["
            Greenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTIO
            N["Eckert_IV"],PARAMETER["False_Easting",0.0],PARAMETER["Fal
            se_Northing",0.0],PARAMETER["Central_Meridian",0.0],UNIT["Me
            ter",1.0]]
```

```
Eckert_VI
            PROJCS["World_Eckert_VI",GEOGCS["GCS_WGS_1984",DATUM["D_WGS_
            1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM["
            Greenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTIO
            N["Eckert_VI"],PARAMETER["False_Easting",0.0],PARAMETER["Fal
            se_Northing",0.0],PARAMETER["Central_Meridian",0.0],UNIT["Me
            ter",1.0]]

Eckert_V
            PROJCS["World_Eckert_V",GEOGCS["GCS_WGS_1984",DATUM["D_WGS_1
            984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM["G
            reenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTION
            ["Eckert_V"],PARAMETER["False_Easting",0.0],PARAMETER["False
            _Northing",0.0],PARAMETER["Central_Meridian",0.0],UNIT["Mete
            r",1.0]]

Equidistant_Conic
            PROJCS["World_Equidistant_Conic",GEOGCS["GCS_WGS_1984",DATUM
            ["D_WGS_1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],
            PRIMEM["Greenwich",0.0],UNIT["Degree",0.0174532925199433]],P
            ROJECTION["Equidistant_Conic"],PARAMETER["False_Easting",0.0
            ],PARAMETER["False_Northing",0.0],PARAMETER["Central_Meridia
            n",0.0],PARAMETER["Standard_Parallel_1",60.0],PARAMETER["Sta
            ndard_Parallel_2",60.0],PARAMETER["Latitude_Of_Origin",0.0],
            UNIT["Meter",1.0]]

Equidistant_Cylindrical
            PROJCS["World_Equidistant_Cylindrical",GEOGCS["GCS_WGS_1984"
            ,DATUM["D_WGS_1984",SPHEROID["WGS_1984",6378137.0,298.257223
            563]],PRIMEM["Greenwich",0.0],UNIT["Degree",0.01745329251994
            33]],PROJECTION["Equidistant_Cylindrical"],PARAMETER["False_
            Easting",0.0],PARAMETER["False_Northing",0.0],PARAMETER["Cen
            tral_Meridian",0.0],PARAMETER["Standard_Parallel_1",60.0],UN
            IT["Meter",1.0]]

Flat_Polar_Quartic
            PROJCS["World_Flat_Polar_Quartic",GEOGCS["GCS_WGS_1984",DATU
            M["D_WGS_1984",SPHEROID["WGS_1984",6378137.0,298.257223563]]
            ,PRIMEM["Greenwich",0.0],UNIT["Degree",0.0174532925199433]],
            PROJECTION["Flat_Polar_Quartic"],PARAMETER["False_Easting",0
            .0],PARAMETER["False_Northing",0.0],PARAMETER["Central_Merid
            ian",0.0],UNIT["Meter",1.0]]

Gall_Stereographic
            PROJCS["World_Gall_Stereographic",GEOGCS["GCS_WGS_1984",DATU
            M["D_WGS_1984",SPHEROID["WGS_1984",6378137.0,298.257223563]]
            ,PRIMEM["Greenwich",0.0],UNIT["Degree",0.0174532925199433]],
            PROJECTION["Gall_Stereographic"],PARAMETER["False_Easting",0
            .0],PARAMETER["False_Northing",0.0],PARAMETER["Central_Merid
            ian",0.0],UNIT["Meter",1.0]]

Gnomonic
            PROJCS["North_Pole_Gnomonic",GEOGCS["GCS_WGS_1984",DATUM["D_
            WGS_1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIM
            EM["Greenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJE
            CTION["Gnomonic"],PARAMETER["False_Easting",0.0],PARAMETER["
            False_Northing",0.0],PARAMETER["Longitude_Of_Center",0.0],PA
            RAMETER["Latitude_Of_Center",90.0],UNIT["Meter",1.0]]
```

```
Hammer_Aitoff
            PROJCS["World_Hammer_Aitoff",GEOGCS["GCS_WGS_1984",DATUM["D_
            WGS_1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIM
            EM["Greenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJE
            CTION["Hammer_Aitoff"],PARAMETER["False_Easting",0.0],PARAME
            TER["False_Northing",0.0],PARAMETER["Central_Meridian",0.0],
            UNIT["Meter",1.0]]

Hotine_Oblique_Mercator_Two_Point_Natural_Origin
            PROJCS["World_Hotine",GEOGCS["GCS_WGS_1984",DATUM["D_WGS_198
            4",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM["Gre
            enwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTION["
            Hotine_Oblique_Mercator_Two_Point_Natural_Origin"],PARAMETER
            ["False_Easting",0.0],PARAMETER["False_Northing",0.0],PARAME
            TER["Latitude_Of_1st_Point",0.0],PARAMETER["Latitude_Of_2nd_
            Point",60.0],PARAMETER["Scale_Factor",1.0],PARAMETER["Longit
            ude_Of_1st_Point",0.0],PARAMETER["Longitude_Of_2nd_Point",60
            .0],PARAMETER["Latitude_Of_Center",40.0],UNIT["Meter",1.0]]

Krovak
            PROJCS["SJTSK_Ferro_Krovak_East_North",GEOGCS["GCS_S_JTSK_Fe
            rro",DATUM["D_S_JTSK",SPHEROID["Bessel_1841",6377397.155,299
            .1528128]],PRIMEM["Ferro",-
            .66666666666667],UNIT["Degree",0.0174532925199433]],PROJECTI
            ON["Krovak"],PARAMETER["False_Easting",0.0],PARAMETER["False
            _Northing",0.0],PARAMETER["Pseudo_Standard_Parallel_1",78.5]
            ,PARAMETER["Scale_Factor",0.9999],PARAMETER["Azimuth",30.288
            13975277778],PARAMETER["Longitude_Of_Center",42.5],PARAMETER
            ["Latitude_Of_Center",49.5],PARAMETER["X_Scale",1.0],PARAMET
            ER["Y_Scale",1.0],PARAMETER["XY_Plane_Rotation",90.0],UNIT["
            Meter",1.0]]

Lambert_Azimuthal_Equal_Area
            PROJCS["North_Pole_Lambert_Azimuthal_Equal_Area",GEOGCS["GCS
            _WGS_1984",DATUM["D_WGS_1984",SPHEROID["WGS_1984",6378137.0,
            298.257223563]],PRIMEM["Greenwich",0.0],UNIT["Degree",0.0174
            532925199433]],PROJECTION["Lambert_Azimuthal_Equal_Area"],PA
            RAMETER["False_Easting",0.0],PARAMETER["False_Northing",0.0]
            ,PARAMETER["Central_Meridian",0.0],PARAMETER["Latitude_Of_Or
            igin",90.0],UNIT["Meter",1.0]]

Lambert_Conformal_Conic
            PROJCS["Africa_Lambert_Conformal_Conic",GEOGCS["GCS_WGS_1984
            ",DATUM["D_WGS_1984",SPHEROID["WGS_1984",6378137.0,298.25722
            3563]],PRIMEM["Greenwich",0.0],UNIT["Degree",0.0174532925199
            433]],PROJECTION["Lambert_Conformal_Conic"],PARAMETER["False
            _Easting",0.0],PARAMETER["False_Northing",0.0],PARAMETER["Ce
            ntral_Meridian",25.0],PARAMETER["Standard_Parallel_1",20.0],
            PARAMETER["Standard_Parallel_2",-
            3.0],PARAMETER["Latitude_Of_Origin",0.0],UNIT["Meter",1.0]]

Loximuthal
            PROJCS["World_Loximuthal",GEOGCS["GCS_WGS_1984",DATUM["D_WGS
            _1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM[
            "Greenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTI
            ON["Loximuthal"],PARAMETER["False_Easting",0.0],PARAMETER["F
            alse_Northing",0.0],PARAMETER["Central_Meridian",0.0],PARAME
            TER["Central_Parallel",40.0],UNIT["Meter",1.0]]
```

```
Mercator
            PROJCS["World_Mercator",GEOGCS["GCS_WGS_1984",DATUM["D_WGS_1
            984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM["G
            reenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTION
            ["Mercator"],PARAMETER["False_Easting",0.0],PARAMETER["False
            _Northing",0.0],PARAMETER["Central_Meridian",0.0],PARAMETER[
            "Standard_Parallel_1",0.0],UNIT["Meter",1.0]]

Miller_Cylindrical
            PROJCS["World_Miller_Cylindrical",GEOGCS["GCS_WGS_1984",DATU
            M["D_WGS_1984",SPHEROID["WGS_1984",6378137.0,298.257223563]]
            ,PRIMEM["Greenwich",0.0],UNIT["Degree",0.0174532925199433]],
            PROJECTION["Miller_Cylindrical"],PARAMETER["False_Easting",0
            .0],PARAMETER["False_Northing",0.0],PARAMETER["Central_Merid
            ian",0.0],UNIT["Meter",1.0]]

Mollweide
            PROJCS["World_Mollweide",GEOGCS["GCS_WGS_1984",DATUM["D_WGS_
            1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM["
            Greenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTIO
            N["Mollweide"],PARAMETER["False_Easting",0.0],PARAMETER["Fal
            se_Northing",0.0],PARAMETER["Central_Meridian",0.0],UNIT["Me
            ter",1.0]]

New_Zealand_Map_Grid
            PROJCS["GD_1949_New_Zealand_Map_Grid",GEOGCS["GCS_New_Zealan
            d_1949",DATUM["D_New_Zealand_1949",SPHEROID["International_1
            924",6378388.0,297.0]],PRIMEM["Greenwich",0.0],UNIT["Degree"
            ,0.0174532925199433]],PROJECTION["New_Zealand_Map_Grid"],PAR
            AMETER["False_Easting",2510000.0],PARAMETER["False_Northing"
            ,6023150.0],PARAMETER["Longitude_Of_Origin",173.0],PARAMETER
            ["Latitude_Of_Origin",-41.0],UNIT["Meter",1.0]]

Orthographic
            PROJCS["North_Pole_Orthographic",GEOGCS["GCS_WGS_1984",DATUM
            ["D_WGS_1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],
            PRIMEM["Greenwich",0.0],UNIT["Degree",0.0174532925199433]],P
            ROJECTION["Orthographic"],PARAMETER["False_Easting",0.0],PAR
            AMETER["False_Northing",0.0],PARAMETER["Longitude_Of_Center"
            ,0.0],PARAMETER["Latitude_Of_Center",90.0],UNIT["Meter",1.0]
            ]

Plate_Carree
            PROJCS["World_Plate_Carree",GEOGCS["GCS_WGS_1984",DATUM["D_W
            GS_1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIME
            M["Greenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJEC
            TION["Plate_Carree"],PARAMETER["False_Easting",0.0],PARAMETE
            R["False_Northing",0.0],PARAMETER["Central_Meridian",0.0],UN
            IT["Meter",1.0]]

Polyconic
            PROJCS["World_Polyconic",GEOGCS["GCS_WGS_1984",DATUM["D_WGS_
            1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM["
            Greenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTIO
            N["Polyconic"],PARAMETER["False_Easting",0.0],PARAMETER["Fal
            se_Northing",0.0],PARAMETER["Central_Meridian",0.0],PARAMETE
            R["Latitude_Of_Origin",0.0],UNIT["Meter",1.0]]
```

```
Quartic_Authalic
           PROJCS["World_Quartic_Authalic",GEOGCS["GCS_WGS_1984",DATUM[
           "D_WGS_1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],P
           RIMEM["Greenwich",0.0],UNIT["Degree",0.0174532925199433]],PR
           OJECTION["Quartic_Authalic"],PARAMETER["False_Easting",0.0],
           PARAMETER["False_Northing",0.0],PARAMETER["Central_Meridian"
           ,0.0],UNIT["Meter",1.0]]

Robinson
           PROJCS["World_Robinson",GEOGCS["GCS_WGS_1984",DATUM["D_WGS_1
           984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM["G
           reenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTION
           ["Robinson"],PARAMETER["False_Easting",0.0],PARAMETER["False
           _Northing",0.0],PARAMETER["Central_Meridian",0.0],UNIT["Mete
           r",1.0]]

Sinusoidal
           PROJCS["World_Sinusoidal",GEOGCS["GCS_WGS_1984",DATUM["D_WGS
           _1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM[
           "Greenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTI
           ON["Sinusoidal"],PARAMETER["False_Easting",0.0],PARAMETER["F
           alse_Northing",0.0],PARAMETER["Central_Meridian",0.0],UNIT["
           Meter",1.0]]

Stereographic
           PROJCS["World_Stereographic",GEOGCS["GCS_WGS_1984",DATUM["D_
           WGS_1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIM
           EM["Greenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJE
           CTION["Stereographic"],PARAMETER["False_Easting",0.0],PARAME
           TER["False_Northing",0.0],PARAMETER["Central_Meridian",0.0],
           PARAMETER["Scale_Factor",1.0],PARAMETER["Latitude_Of_Origin"
           ,0.0],UNIT["Meter",1.0]]

Times
           PROJCS["World_Times",GEOGCS["GCS_WGS_1984",DATUM["D_WGS_1984
           ",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM["Gree
           nwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTION["T
           imes"],PARAMETER["False_Easting",0.0],PARAMETER["False_North
           ing",0.0],PARAMETER["Central_Meridian",0.0],UNIT["Meter",1.0
           ]]

Transverse_Mercator
           PROJCS["WGS_1984_UTM_Zone_2N",GEOGCS["GCS_WGS_1984",DATUM["D
           _WGS_1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRI
           MEM["Greenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJ
           ECTION["Transverse_Mercator"],PARAMETER["False_Easting",5000
           00.0],PARAMETER["False_Northing",0.0],PARAMETER["Central_Mer
           idian",-
           171.0],PARAMETER["Scale_Factor",0.9996],PARAMETER["Latitude_
           Of_Origin",0.0],UNIT["Meter",1.0]]
```

```
Two_Point_Equidistant
           PROJCS["World_Two_Point_Equidistant",GEOGCS["GCS_WGS_1984",D
           ATUM["D_WGS_1984",SPHEROID["WGS_1984",6378137.0,298.25722356
           3]],PRIMEM["Greenwich",0.0],UNIT["Degree",0.0174532925199433
           ]],PROJECTION["Two_Point_Equidistant"],PARAMETER["False_East
           ing",0.0],PARAMETER["False_Northing",0.0],PARAMETER["Latitud
           e_Of_1st_Point",0.0],PARAMETER["Latitude_Of_2nd_Point",60.0]
           ,PARAMETER["Longitude_Of_1st_Point",0.0],PARAMETER["Longitud
           e_Of_2nd_Point",60.0],UNIT["Meter",1.0]]

Van_der_Grinten_I
           PROJCS["World_Van_der_Grinten_I",GEOGCS["GCS_WGS_1984",DATUM
           ["D_WGS_1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],
           PRIMEM["Greenwich",0.0],UNIT["Degree",0.0174532925199433]],P
           ROJECTION["Van_der_Grinten_I"],PARAMETER["False_Easting",0.0
           ],PARAMETER["False_Northing",0.0],PARAMETER["Central_Meridia
           n",0.0],UNIT["Meter",1.0]]

Vertical_Near_Side_Perspective
           PROJCS["Sphere_Vertical_Perspective",GEOGCS["GCS_WGS_1984",D
           ATUM["D_WGS_1984",SPHEROID["WGS_1984",6378137.0,298.25722356
           3]],PRIMEM["Greenwich",0.0],UNIT["Degree",0.0174532925199433
           ]],PROJECTION["Vertical_Near_Side_Perspective"],PARAMETER["F
           alse_Easting",0.0],PARAMETER["False_Northing",0.0],PARAMETER
           ["Longitude_Of_Center",0.0],PARAMETER["Latitude_Of_Center",0
           .0],PARAMETER["Height",35800000.0],UNIT["Meter",1.0]]

Winkel_II
           PROJCS["World_Winkel_II",GEOGCS["GCS_WGS_1984",DATUM["D_WGS_
           1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM["
           Greenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTIO
           N["Winkel_II"],PARAMETER["False_Easting",0.0],PARAMETER["Fal
           se_Northing",0.0],PARAMETER["Central_Meridian",0.0],PARAMETE
           R["Standard_Parallel_1",50.4597762521898],UNIT["Meter",1.0]]

Winkel_I
           PROJCS["World_Winkel_I",GEOGCS["GCS_WGS_1984",DATUM["D_WGS_1
           984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM["G
           reenwich",0.0],UNIT["Degree",0.0174532925199433]],PROJECTION
           ["Winkel_I"],PARAMETER["False_Easting",0.0],PARAMETER["False
           _Northing",0.0],PARAMETER["Central_Meridian",0.0],PARAMETER[
           "Standard_Parallel_1",50.4597762521898],UNIT["Meter",1.0]]

Winkel_Tripel
           PROJCS["World_Winkel_Tripel_NGS",GEOGCS["GCS_WGS_1984",DATUM
           ["D_WGS_1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],
           PRIMEM["Greenwich",0.0],UNIT["Degree",0.0174532925199433]],P
           ROJECTION["Winkel_Tripel"],PARAMETER["False_Easting",0.0],PA
           RAMETER["False_Northing",0.0],PARAMETER["Central_Meridian",0
           .0],PARAMETER["Standard_Parallel_1",50.467],UNIT["Meter",1.0
           ]]
```

Appendix D: Using S-Expressions to Specify Grid Extraction

This appendix describes a proposed standard for specifying GRDSpec values, the abstract datatype used by the Grid DataBlade to control what information is extracted from a GRDValue. The standard is based on S-expressions, which are a hierarchical text based data structure in the style of LISP.

Briefly, an S-expression consists of a set of zero or more terms enclosed by a pair of parenthesis. Each term is either an atom or another S-expression. An atom can be any string of characters that does not include a parenthesis, white space, or a quote. Alternatively, an atom can be a string of characters surrounded by double quotes (“)[8].

Terms in S-Expressions Understood by the Grid DataBlade

The Grid DataBlade understands the following terms (bold values are to be interpreted as literals):

(fill_holes)
(**translation** x y z t)
(**affine_transformation** $r_{1,1}$ $r_{1,2}$ $r_{1,3}$ $r_{1,4}$ … $r_{4,1}$ $r_{4,2}$ $r_{4,3}$ $r_{4,4}$)
(**dim_sizes** dim_1_size dim_2_size dim_3_size dim_4_size)
(**dim_names** dim_1_name dim_2_name dim_3_name dim_4_name)
(**coord_wraps_in (**axisName minValue maxValue) (…))
(**nonuniform** dim_i_name position1 position2 … position n)
(**linear_interpolation** dimi_name …)
(**variables** var_1_name … var_v_name)
(**srtext** text)

Notes:

- The **srtext**, **translation**, and **affine_transformation** terms have descriptions identical to the identically named attributes in a GIEF file. The one exception is that **srtext** can use (@) in place of double quotes (“) if desired. That simplifies embedding the srtext inside of a string in a scripting file.
- The **dim_names** term is a list of the names of the dimensions (in most major to least major order) of the grid. This can be used to replace the dimension names in the exported file.
- The **variables** term lists the variables to be extracted from the source grid. These names must be drawn from the set of names of fields actually in the grid.
- The **dim_sizes** term specifies the number of samples in each dimension.
- The **nonuniform** terms (there may be one for each dimension) allow one to list the values along a particular dimension where values should be sampled.

[8] The Grid DataBlade functions will transform backticks (`) to double quotes to provide an additional level of quoting when using command line utilities such as dbaccess.

- The **fill_holes** term indicates that the extract should attempt to patch holes in the data with adjacent values. The resulting extracted grid may look more complete without the holes, but the values at those positions will not reflect the data that would have been there if there had not been holes in the data. Note: **fill_holes** operates by replicating non-missing data in place of missing data, prior to linear interpolation. It has no effect when linear interpolation is not in effect.
- The **linear_interpolation** term is used to control whether nearest neighbours or linear interpolation is used to compute a value for a sample that is not exactly on a grid point. If a particular dim_name is listed in the term, the corresponding dimension will be linearly interpolated.
- The **coord_wraps_in** term is used to denote which world dimensions wrap-around, and at what values they wrap-around. The allowed values for the axisName are *x*, *y*, *z*, and *t*. For example, geographic coordinates might have the term: *(coord_wraps_in (x -180 180))*. To denote that, longitude wraps around.

In general, the order of the terms is arbitrary, with the following exceptions:

- The **dim_sizes** term must come before the **dim_names** term.
- The **dim_names** term must come before any term that references a dim_name, such as **nonuniform** terms or a **linear_interpolation** term.

Example of an S-Expression

```
 ((srtext
'PROJCS["Madrid 1870 (Madrid) / Spain",GEOGCS["Madrid 1870 Madrid)",
DATUM["Madrid_1870",SPHEROID["Struve1860",6378298.3,294.73]],
PRIMEM["Madrid",-3.68793888888889],UNIT["degree",0.0174532925199433]],
PROJECTION["Lambert_Conformal_Conic_1SP"],
PARAMETER["latitude_of_origin",40],PARAMETER["central_meridian",0],
PARAMETER["scale_factor",0.9988085293],PARAMETER["false_easting",600000],
PARAMETER["false_northing",600000],UNIT["metre",1]]')
(dim_sizes 50 30 4 3)
(dim_names column row level time)
(translation 10 30 5 1000)
(variables air_temp)
 (affine_transform  1 0 0 0
                    0 1 0 0
                    0 0 1 0
                    0 0 0 1)
(nonuniform time 3.0 4.5 9.2)
(nonuniform level 100 1000 2000 3000)
                  )
```

Alternatively, employing @ symbols instead of double quotes:

```
((srtext
'PROJCS[@Madrid 1870 (Madrid) / Spain@,GEOGCS[@Madrid 1870 Madrid)@,
DATUM[@Madrid_1870@,SPHEROID[@Struve1860@,6378298.3,294.73]],
PRIMEM[@Madrid@,-3.68793888888889],UNIT[@degree@,0.0174532925199433]],
PROJECTION[@Lambert_Conformal_Conic_1SP@],
PARAMETER[@latitude_of_origin@,40],PARAMETER[@central_meridian@,0],
PARAMETER[@scale_factor@,0.9988085293],PARAMETER[@false_easting@,60000
0],
PARAMETER[@false_northing@,600000],UNIT[@metre@,1]]')
(dim_names column row level time)
(translation 10 30 5 1000)
(variables air_temp)
(dim_sizes 50 30 4 3)
(affine_transform  1 0 0 0
                   0 1 0 0
                   0 0 1 0
                   0 0 0 1)
(nonuniform time 3.0 4.5 9.2)
(nonuniform level 100 1000 2000 3000)
                )
```

Formal S-Expression Definition

Formally, S-expressions are specified as either:

- **Tokens**. The following is a specification of how tokens are built out of characters.

  <token> ---> <identifier> | <boolean> | <number> | <string> | (|) | ' | , | .
  <delimiter> ---> <whitespace> | (|) | " | ;
  <whitespace> ---> <space or newline>
  <comment> ---> ; <all subsequent characters up to a line break or EOF>
  <atmosphere> ---> <whitespace> | <comment>
  <intertoken space> ---> <atmosphere>*
  <identifier> ---> <initial> <subsequent>*
  <initial> ---> <letter> | <special initial>
  <letter> ---> lower and upper-case letters: a-z A-Z
  <special initial> ---> ! | $ | % | & | * | / | : | < | = | > | ? | ^ | _ | ~
  <subsequent> ---> <initial> | <digit> | <special subsequent>
  <special subsequent> ---> + | - | . | @
  <digit> ---> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
  <boolean> ---> #t | #f
  <number> ---> whatever is accepted by sscanf with %lf and %ld formats.
  <string> ---> " <string element>* "
  <string element> ---> <any character other than " or \> | \" | \\

Note: <intertoken space> may occur on either side of any token, but not within a token. Tokens which require implicit termination (identifiers, numbers, and dot) may be terminated by any <delimiter>, but not necessarily by anything else.

- **S-expressions**. These are made of tokens in the following way:

  <expression> ---> <identifier> | <literal> | <compound>
  <literal> ---> <boolean> | <number> | <string>
  <compound> ---> (<expression> <expression>*)

Appendix E: Error Messages

There are two classes of error messages that may be generated by the Grid DataBlade, namely, user errors and program failure errors. User errors are those that an application implementor or database user may inadvertently cause. Program failure errors reflect either programming errors in the DataBlade or a corrupted database, and are error messages that the application implementor and database user should never be able to bring about. Both classes of error messages, along with explanations, are described in this section.

User Error Messages

The Grid DataBlade permits an application to create a grid that is larger than can be represented in a 32-bit address space. Attempts to extract such a grid in its entirety would cause memory allocation failures (at the very least), and so the Grid DataBlade imposes some reasonable limits on sizes of grids that can be used in certain operations. In the text below, GRDValues larger than 2GB will be described as "too large" (for the operation).

The user error messages are as follows:

attempt to convert too large a grid to text: The server was requested to convert a grid that was too large into text.
attempt to pass in too large a grid: The client program tried to pass in a GRDValue that was too large.
attempt to return too large a grid: The client program tried to fetch a grid that was too large.
attempt to extract too large a grid: The client program tried to extract a grid that was too large.
attempt to update with too large a grid: The client program tried to update a grid (in place) with a new grid that was too large.

update of GRDValue with NULL not permitted: The client program tried to update a GRDValue with a null. This operation is considered illegal since updates of GRDValues have a modify rather than a replace semantics.

update of null GRDValue not permitted: The client program tried to update a GRDValue that was previously null. A null GRDValue can occur if a user performs an insert with a column list that does not include the GRDValue column. This is treated as illegal because the current update scheme does an in-place update of the underlying large object and nulls have no underlying large object.

incompatible GRDValue used in update: In order for one GRDValue to be used to update another, the two GRDValues must have the same number of dimensions, the same basis vectors, the same projections, and all the fields in the source GRDValue must be present in the destination GRDValue. This message indicates that one of more of the above requirements was not satisfied.

BuildLargeObject: **mi_lo_create failed**: This error indicates that that the smart blob space being used to store the GRDValue is too small. If it occurs, your Informix database administrator should review the size of your data and how much space is actually reserved in the associated smart blob space.

BuildLargeObject: **mi_lo_write failed**: See the above message.

attempt to copy large object failed: See the above message.

attempt to create large object failed: See the above message.

Program Failure Error Messages

The first class of program failure error messages indicates a memory problem inside the server. This can be caused by errors in the server code, errors in the Grid DataBlade, or errors in another party's DataBlade (the server operates in a threaded memory model without internal protection, so one DataBlade can interfere unintentionally with another, or with the server itself). It can also be caused by attempting to execute a query that requires more memory than available. These memory-related error messages are:

mi_alloc of size %d failed: Memory is corrupted or unavailable; the database server may need to be restarted.

GRDCalloc of size %d,%d failed: See the above message.

mi_dalloc(%s,PER_COMMAND) failed: See the above message.

The second class of program failure error messages is "inaccessible", since there is no known way to generate them. Their presence may indicate a corrupted database (for instance, if a hard disk failed or was not mounted), or an application program that has managed to corrupt a GRDValue before it could be passed to the server. These "inaccessible" error messages are:

'lots of writes' case not implemented yet:
BuildLargeObject: both lengths 0, programmer error

attempt to malloc a negative amount of memory
attempt to open LO failed
attempt to open connection failed
attempt to read very large LO in small way
bad symbol parsing
attempt to update GRDValue not in large objectmi_lo_close failed
mi_lo_open failed
mi_lo_read failed
mi_lo_read_withseek failed
mi_lo_readwithseek failed
mi_lo_writewithseek failed
not currently implemented
partialBlobRead exceeded length of memory blob
programmer error: bad blob storage type
unable to close large object
unable to open connection to ius database=%s user=%s password=%s
unable to reopen large object after creating it
unable to stat LO
unable to stat_size LO
unable to write to large object